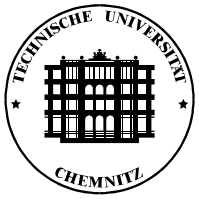
ISYM - Information Systems & Management
Chemnitz University of Technology
Prof. Dr. Peter Loos and Prof. Dr. Bernd Stöckert



Paper 2

P. Loos, P. Fettke

Aspekte des Wissensmanagements in der Software-Entwicklung am Beispiel von V-Modell und Extreme Programming

2001

Working Papers of the Research Group Information Systems & Management

Publisher:

Prof. Dr. Peter Loos
Prof. Dr. Bernd Stöckert
Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Information Systems & Management
D-09107 Chemnitz, Germany

http://www.isym.tu-chemnitz.de

ISSN 1617-6324 (printed version)
ISSN 1617-6332 (Internet version)

Management Summary

Der in der Literatur verwendete Wissensbegriff ist vielschichtig. In dieser Arbeit werden sechs Facetten des Begriffes Wissen aus Sicht der Software-Entwicklung herausgearbeitet: Träger, Güte, Ausdrücklichkeit, Sprachbezug, Abstraktionsebene und Inhalt. Während traditionelle, schwergewichtige Software-Entwicklungsprozesse primär explizites und formal-sprachliches Wissen fokussieren, das sich bspw. in Anforderungsdokumenten oder Quelltexten widerspiegelt, betonen leichtgewichtige Prozesse neben dem formal-sprachlichen Wissen auch implizites und nicht-sprachliches Wissen, das durch Konzepte wie Pair-Progamming oder Planspiele bei der Projektplanung stimuliert wird. Bei einer Untersuchung des V-Modells und des Extreme Programming werden weitere Unterschiede alternativer Vorgehensweisen bei der Software-Entwicklung aus Sicht des Wissensmanagements dargestellt. Abgeschlossen wird die Arbeit mit einer Zusammenstellung spezieller Anforderungen an ein Wissensmanagement in der Software-Entwicklung.

Keywords: Wissen, Erfahrungsdatenbanken, Software-Management, Software-Engineering, Ziele, Anforderungen, leichtgewichtige Prozesse, schwergewichtige Prozesse

Der Aufsatz ist auch erschienen in der virtuellen Festschrift anlässlich des 60. Geburtstags von Herrn Prof. Dr. Dr. h.c. A.-W. Scheer (siehe http://www.aws60.de).

Authors

Prof. Dr. Peter Loos, Peter Fettke
Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Information Systems & Management
D-09107 Chemnitz, Germany
Phone: +49/371/531-4375, Fax: -4376
E-Mail: {loos|peter.fettke}@isym.tu-chemnitz.de

Inhaltsverzeichnis

The central question of how to improve the software art centers, as it always has, on people. [...] Great designs come from great designers. Software construction is a creative *process. Sound methodology can empower and liberate the creative mind; it cannot enflame or inspire the drudge.*
Frederick P. Brooks, 1995, S. 202

I've always believed that one of the best ways to learn is by a process of trial and error.
Donald E. Knuth, 1992, S. 28

Computer Programming is a human activity.
Gerald M. Weinberg, 1971, S. 3

1 Software-Entwicklung als wissensintensiver Prozess

In den letzten Jahren hat das Thema Wissensmanagement (WM) sowohl in der Theorie als auch in der Praxis große Aufmerksamkeit erfahren.[1] WM behandelt im Wesentlichen die systematische Planung, Steuerung und Kontrolle aller Prozesse einer Unternehmung im Hinblick auf ihren Bedarf an Wissen und ihre Erzeugung von Wissen.[2] Aus dieser Perspektive kann Wissen als ein eigenständiger Produktionsfaktor verstanden werden,[3] wobei die Aufgaben des WM alle funktionalen Bereiche eines Unternehmens tangieren.

Gleichzeitig nimmt die Software-Entwicklung innerhalb des gesamten volkswirtschaftlichen Wertschöpfungsbereichs eine zunehmende Rolle ein.[4] Vor diesem Hintergrund stellt sich die Frage, wie die Prozesse der Software-Entwicklung zu WM in Verbindung stehen: Menschen bilden den operativen Bereich eines Software entwickelnden Unternehmens. Andere Produktionsfaktoren wie Computer, Netzwerktechnik, CD-Pressen u. ä. nehmen den Rang von Fertigungshilfsmitteln ein. Weitere maschinelle bzw. materielle Produktionsmittel, die als kritische Erfolgsfaktoren zur Erstellung von Software einzuschätzen sind, existieren nicht. Dies zeigt, dass die Hauptprozesse der Software-Entwicklung aus Prozessen der Verarbeitung von Daten, Informationen sowie Wissen bestehen, die im wesentlichen von menschlichen Akteuren ausgeführt werden. In diesem Kontext spricht Pawlowsky von einer „Entmaterialisierung von Wertschöpfungsketten", d. h. „Gebäude, Maschinen und Produkte werden durch Gedanken ersetzt"[5]. Diese Eigenschaft trifft auf Unternehmen der Software-Entwicklung in besonderer Weise zu. Die Erfahrungen, Kompetenzen und Fähigkeiten von System-Analytikern, Software-Architekten, Programmieren, Testern usw. haben einen entscheidenden Einfluss auf die Qualität der Software-Entwicklungsprozesse und damit auf die entwickelten Software-Produkte.[6] Allerdings ist Wissen nicht nur ein dominanter Produktionsfaktor. Die von einer Software-Organisation entwickelten Software-Programme können ebenso als Wissen interpretiert werden, da Software-Produkte in hohem Umfang Gestaltungsmöglichkeiten und -grenzen

[1] Vgl. Igl, Lehner 2000; Schindler 2001, S. 1-6; Seidel, Lehner 2000.

[2] Vgl. Raub 2000, S. 3543.

[3] Vgl. Scheer 1998, S. 3.

[4] Vgl. Friedewald et al. 2001, S. 82-85.

[5] Pawlowsky 1998, S. 15.

[6] Vgl. Solingen et al. 2000.

der betrieblichen Organisation widerspiegeln. Aus diesem Grund sollte Software-Entwicklung als eine wissensintensive betriebliche Aufgabe verstanden werden, die nicht nur aus der Sicht des Software-Engineering, sondern auch aus der Sicht des WM zu behandeln ist.

Betrachtet man die einschlägige Literatur, ist festzustellen, dass zwar eine Vielzahl an Werken vorliegt, die den breiteren Themenkomplexen Wissensmanagement oder Software-Entwicklung zuzurechnen sind. Allerdings ist Literatur, die explizit das Thema Wissensmanagement in der Software-Entwicklung aufgreift, spärlich vorhanden. Dieser Artikel soll einen Beitrag zur Milderung dieses Defizits leisten: Etablierte und aktuelle Vorgehensmodelle der Software-Entwicklung werden unter dem Aspekte des WM näher untersucht.

Die Untersuchung wird zunächst in Kapitel 2 den Begriff Wissen aus Sicht der Software-Entwicklung reflektieren und verschiedene Bausteine eines WM kurz beschreiben. Im anschließenden Kapitel 3 werden die unterschiedlichen Zielstellungen eines WM in der Software-Entwicklung näher herausgearbeitet. Kapitel 4 fokussiert Aspekte des WM im V-Modell und bei Extreme Programming. Abschließend werden in Kapitel 5 verschiedene Schlussfolgerungen gezogen, die als Anforderungen an ein WM in der Software-Entwicklung zu interpretieren sind.

2 Wissen und Wissensmanagement aus Sicht der Software-Entwicklung

2.1 Zum Wissensbegriff

Die Auffassungen, was unter dem Begriff Wissen zu verstehen ist, sind heterogen. Schon Philosophie und Wissenschaftstheorie im allgemeinen sowie die Erkenntnistheorie im besondern haben in ihrer langen Tradition zahlreiche Ansätze entwickelt, um zu klären, was unter Wissen zu verstehen ist. Entscheidend ist dabei die Frage, ob objektives Wissen existiert, wie dieses gewonnen werden kann und wie dieses sich vom bloßen Glauben und Meinen abgegrenzt.[7]

Ebenso existieren eine Reihe von Definitionsansätzen, die insbesondere die Begriffe Wissen und Information in Beziehung setzen und voneinander abgrenzen. Einerseits gibt es Definitionsversuche, bei denen der Begriff Information unter Rückgriff auf den Begriff Wissen eingeführt wird: „Information ist zweckorientiertes Wissen, wobei der Zweck in der Vorbereitung des Handelns liegt.“[8] Andererseits beschreiten andere Arbeiten den Weg, Wissen mit Bezug auf den Begriff Information zu definieren. Beispielsweise stellen Maier und Klosa die plakative Formel „Wissen = Information + Kontext“ auf.[9] Andere Autoren betonen die Wichtigkeit, Wissen nicht nur im Hinblick auf eine einzelne Person, sondern auch auf eine Gruppe von Menschen in einer Organisationen zu verstehen.[10]

Weitere Arbeiten stellen bei der Definition des Wissensbegriffs einen Bezug zur Semiotik her. Ausgangspunkt der Überlegung ist die Frage, wie Nachrichten zwischen Sendern und Empfängern ausgetauscht werden. Die Semiotik betrachtet Nachrichten auf einer syntaktischen,

---

[7] Vgl. Seiffert, Radnitzky 1992, S. 387-391.

[8] Gemünden 1993, Sp. 1725.

[9] Vgl. Maier, Klosa 1999b, S. 3.

[10] Vgl. Willke 1998, S. 41.

einer semantischen sowie einer pragmatischen Ebene. In Anlehnung an die semiotischen Ebenen werden die Begriffe Zeichen, Daten und Information eingeführt. Bei diesen Ansätzen wird Wissen als das Verstehen, Verarbeiten und Bewerten von Informationen verstanden.[11] Dabei können die Begriffsebenen oft im Einzelfall nicht sauber unterschieden werden.[12] Ebenso wird der Wissensbegriff innerhalb der Künstlichen Intelligenz im Zusammenhang mit Systemen verwendet, bei denen „das Fachwissen über ein Anwendungsgebiet [...] explizit und unabhängig vom allgemeinen Problemlösungswissen dargestellt wird".[13]

Dagegen besteht aus sozialpädagogischer Sicht die Auffassung, dass Wissen die Basis ist, um anspruchsvollere Stufen im Lernprozess zu erreichen. Das Wissen um bestimmte Sachverhalte ist die Voraussetzung, um diese zu verstehen. Verstehen wiederum kann als eine Bedingung für die Bewertung von Konzepten verstanden werden.[14]

Die bisher präsentierte Übersicht über verschiedene Wissensbegriffe erhebt keinen Anspruch auf Vollständigkeit, sondern stellt exemplarisch verschiedene Auffassungen gegenüber, um die Bedeutungsvielfalt und Interpretationsspielräume des Wissensbegriffs zu verdeutlichen.[15] Die Zielsetzung dieser Arbeit macht eine scharfe Abgrenzung des Begriffes Wissen nicht erforderlich. Um diesem Begriff dennoch die notwendige Aussagekraft zu verleihen, wird er im Folgenden einer morphologischen Betrachtung unterzogen, die unterschiedliche Facetten des Begriffs beleuchtet. Doch zunächst wird der Wissensbegriff in einer weiten Auffassung kurz und allgemein umrissen:

Eine Nachricht ist eine Mitteilung eines Menschen an einen anderen Menschen, deren Struktur durch eine Sprache im weitesten Sinn festgelegt ist. Im weitesten Sinn heißt, dass sowohl formale, semi-formale und natürliche Sprachen als auch non-verbale Sprachen wie bspw. Rauchzeichen umfasst werden sollen. Daten sind automatisiert verarbeitbare Nachrichten. Der Begriff Kommunikation bezeichnet das Austauschen von Nachrichten. Eine Information ist das Ergebnis von Kommunikation und kann interpretiert werden als die Teilmenge von Wissen, die mittels Nachrichten zwischen Menschen ausgetauscht worden ist. Der Begriff Wissen soll als Sammelbegriff für die Wahrnehmung, die Erfahrung, die Kenntnisse, die Intuition, die Vermutungen, den Glauben, die Bildung und die Urteilskraft von Menschen verstanden werden. Die gegebene Umschreibung der Begriffe Nachricht, Daten, Kommunikation, Information und Wissen erfolgte in Anlehnung an eine allgemeine Terminologie der Software-Technik.[16] Der so umgrenzte allgemeine Wissensbegriff wird im Folgenden darüber hinaus aus sechs verschiedenen Blickrichtungen, sogenannten Facetten, im Hinblick auf ein WM stärker profiliert.[17] Dabei wird jede Facette zunächst allgemein erläutert und anschließend an Hand von Beispielen im Kontext der Software-Entwicklung diskutiert.

---

[11] Vgl. Eulgem 1998, S. 15-24; Rehäuser, Krcmar 1996, S. 5f.

[12] Vgl. Frank, Schauer 2001.

[13] Kurbel 1992, S. 18.

[14] Vgl. Huitt 2000.

[15] Lehner, Hildebrand, Maier 1995, S. 165-272, geben einen ausführlichen Überblick über die Begriffe Daten, Information sowie Wissen aus Sicht der Betriebswirtschaftslehre, Informatik sowie Wirtschaftsinformatik.

[16] Vgl. Barkow et al. 1989; Hesse et al. 1994; Heinrich, Roithmayr 1995 S. 258.

[17] Vgl. auch Bea 2000, S. 362f., der 4 Facetten unterscheidet.

1. Facette: Träger

Der Wissensbegriff kann in Hinblick auf verschiedene Träger des Wissens differenziert werden. Die Facette kann grob in die Bereiche menschliche und technische Träger von Wissen unterteilt werden. Bei einer feineren Betrachtung können menschliche Wissensträger sowohl einzelne Individuen, eine Gruppe, eine Organisation sowie ein Unternehmens-Netzwerk darstellen.

Beispiele: 1. Individuum: Der Software-Architekt Herr M. (Wissensträger) ist der einzige im Projektteam, der schon in vergangenen Entwicklungsprojekten mit dem Unified Modeling Language-Modellierungswerkzeug Y gearbeitet und entsprechende Erfahrungen gesammelt hat. 2. Gruppe: Die Arbeitsabläufe in einer Verwaltung sollen informationstechnisch unterstützt werden. Dazu wird das Wissen der einzelnen Sachbearbeiter (Wissensträger) durch Systemanalytiker mit Hilfe von Interview-Techniken erhoben. 3. Organisation: Das Unternehmen U (Wissensträger) entwickelt seit 15 Jahren betriebliche Standardsoftware für die Produktionsplanung- und -steuerung und ist daher mit den informationslogistischen Prozessen in Fertigungsbetrieben bestens vertraut. 4. Netzwerk: Das zuvor beschriebene Unternehmen arbeitet mit verschiedenen Vertragspartnern zusammen, die gemeinsam als Wissensträger interpretiert werden können: Bspw. existieren mit verschiedenen Partnern feste Beziehungen, die für die Beratung und Einführung des Standardsoftware-Systems abgeschlossen worden sind. Das Zusammenspiel zwischen den verschiedenen Organisationen funktioniert inzwischen reibungslos, obgleich es zu Beginn einer neuen Partnerschaft stets zu starken Friktionen kommt. 5. Technisches System: Das zuvor beschriebene Unternehmen U verfügt über ein Anwendungssystem-Framework (Wissensträger), das inzwischen als Basis für alle Kundenprojekte verwendet wird.

2. Facette: Güte

Diese Facette beschreibt, mit welcher methodischen Stringenz das Wissen geschaffen wurde. Zum einen kann Wissen subjektiv sein. Subjektives Wissen ist nicht exakt fundiert. Objektives Wissen dagegen ist mit einer anerkannten Methode erzeugt worden, ist systematischer Natur und steht in einem Begründungszusammenhang. Auch wenn die beiden genannten Kategorien eher als die Pole eines Kontinuums verstanden werden können, werden sie in dieser Arbeit als zwei diskrete Klassen eingeführt.

Beispiele: 1. Subjektives Wissen: Die Projektleiterin Frau B. hat die Erfahrung gemacht, dass die Funktionalität einer marktüblichen Tabellenkalkulation T für die Planung, Steuerung und Überwachung ihrer Projekte ein effizientes Werkzeug ist. Ein professionelles und dediziertes Projektmanagementwerkzeug P erlaubt ihr zwar die exaktere Abbildung von Reihenfolgebedingungen von Aktivitäten, allerdings ist die Bedienung von P um ein Vielfaches komplizierter und komplexer. Daher zieht sie das Werkzeug T ohne nähere Untersuchung dem Werkzeug P vor und nutzt dieses für das Projektmanagement. 2. Objektives Wissen: Das Prinzip des Information-Hiding ist ein in Theorie und Praxis anerkanntes Prinzip zur Strukturierung von Software-Systemen.

3. Facette: Ausdrücklichkeit

Grundsätzlich kann zwischen implizitem und explizitem Wissen unterschieden werden.[18] Beide Dimensionen sollen unter dem Begriff Ausdrücklichkeit zusammengefasst werden. Explizites Wissen ist direkt und unmittelbar erkennbar und für sich als solches kommunizierbar und verständlich. Es kann leicht sprachlich erfasst werden. Der Vorgang der Explikation von implizitem Wissen ist oft nur schwer oder gar nicht möglich, so dass implizites Wissen nur meta-sprachlich erfasst werden kann, indem explizites Wissen über implizites Wissen formuliert wird.

Beispiele: 1. Implizites Wissen: Eine wichtige Tätigkeit im Rahmen der Systemanalyse ist die Identifikation von fachlichen Objekten und die Verteilung von Verhalten auf diese Objekte. System-Analytiker mit Berufserfahrung verfügen über entsprechende Fähigkeiten, den Prozess der Systemanalyse erfolgreich zu gestalten. Es sei darauf hingewiesen, dass erfahrene System-Analytiker kaum in der Lage sind, ihr Vorgehen zur Objektfindung algorithmisch zu beschreiben. Ebenso kennt die Wissenschaft nur allgemeine Heuristiken, die den Prozess der Objektfindung leiten können. 2. Explizites Wissen: Im Rahmen der Analyse eines Patienteninformationssystems müssen diverse rechtliche Rahmenbedingungen wie die ärztliche Schweigepflicht, das Datenschutzrecht u. ä. berücksichtigt werden. Bei den entsprechenden rechtlichen Vorschriften handelt es sich um explizites Wissen.

4. Facette: Sprachbezug

Sprachliches Wissen ist in einer Sprache formuliert. Bei formal-sprachlichem Wissen ist die verwendete Sprache formal definiert, anders dagegen bei natürlich-sprachlichen Ausdrücken. Bei nicht-sprachlichem Wissen liegt das Wissen in einer Form vor, die nicht sprachlich fassbar ist. Nicht-sprachliches Wissen ist Wissen, das nicht natürlich-sprachlich, formal-sprachlich oder durch andere Zeichen definiert ist.

Beispiele: 1. Nicht-sprachliches Wissen: Erich Gamma hat im Rahmen des ET++ Projektes an der Universität Zürich die Erfahrung gemacht, dass bestimmte, immer wieder verwendete Entwürfe bei der Software-Entwicklung auf einer abstrakten Ebene Gemeinsamkeiten aufweisen. Bevor Gamma dieses Wissen in Form von Entwurfsmustern in seiner Dissertation[19] versprachlicht hat, verfügte Gamma über nicht-sprachliches Wissen zu ausgezeichneten Systementwürfen. 2. Natürlich-sprachliches Wissen: Im Rahmen des Systementwurfs wird das Musterhandbuch von Gamma et al.[20] verwendet. Dort ist das Muster Composite beschrieben. Die Beschreibung erläutert allgemeinsprachlich den Zweck des Musters. Darüber hinaus werden Anwendungsbereiche, verschiedene Trade-Offs des Musters sowie Aspekte, die bei der Implementierung zu berücksichtigen sind, normalsprachlich erläutert. 3. Formalsprachliches Wissen: Das erwähnte Musterhandbuch verwendet darüber hinaus ebenso eine graphische Modellierungssprache zur Beschreibung von Klassenstrukturen und Objektinteraktionen, die einer explizit definierten Notation gehorchen. Ferner werden entsprechende Implementierungsbeispiele des Musters in der Programmiersprache C++ dargelegt. Die Klassenstrukturen sowie die exemplarischen Implementierungen können als formal-sprachliches Wissen verstanden werden.

---

[18] Nonaka, Takeuchi 1995, S. 8-11, bezeichnen implizites Wissen als „tacit knowledge“.

[19] Vgl. Gamma 1992.

[20] Gamma et al. 1995.

5. Facette Abstraktionsebene

Wissen kann hinsichtlich verschiedener Abstraktionsgrade klassifiziert werden. Wissen auf unterster Abstraktionsstufe ist Wissen über konkrete Sachverhalte, es handelt sich hierbei um Faktenwissen. Konzeptuelles Wissen steht auf der zweiten Abstraktionsstufe und kann als Wissen über Sachwissen verstanden werden. Auf einer dritten Abstraktionsstufe kann von Wissen über konzeptuelles Wissen gesprochen werden, das auch als Meta-Wissen bezeichnet werden kann.[21]

Beispiele: 1. Sachwissen: Im Rahmen einer frühen Phase der Systementwicklung werden die Geschäftsprozesse eines Fertigungsunternehmens F in ihrem Ist-Zustand erhoben. Das Ergebnis der Analyse ist Sachwissen. 2. Konzeptuelles Wissen: Ein Unternehmen plant ein Anwendungssystem-Framework für Fertigungsindustrieunternehmen zu erstellen. Es führt eine Domänenanalyse in diesem Anwendungsbereich durch, um so konzeptuelles Wissen über allgemeine Geschäftsprozesse in Fertigungsunternehmen zu erheben. 3. Meta-Wissen: Dasselbe Unternehmen untersucht ebenfalls Geschäftsbereiche angrenzender Industriebereiche, um so Aufschluss darüber zu erhalten, wie sich die Geschäftsprozesse eines Fertigungsunternehmens von denen in Unternehmen anderer Wirtschaftszweige (Banken, Versicherungen oder Handel) unterscheiden. Am Ende der Untersuchungen entsteht ein Model (Meta-Wissen), das erlaubt, Geschäftsprozesse in Fertigungsunternehmen und anderen Wirtschaftszeigen auf Basis ausgewählter Kriterien und spezifischer Merkmale zu beschreiben.

6. Facette: Sachbezug

Neben den oben aufgeführten allgemeinen Facetten des Wissensbegriffs ist aus der Sicht der Software-Entwicklung eine weitere Facette von Interesse: Wissen kann hinsichtlich seines Sachbezugs inhaltlich in Domänen-, Technologie- und Projektwissen differenziert werden.

Im Kontext dieser Arbeit wird von der Software-Entwicklung in Wirtschaft und Verwaltung ausgegangen. Hintergrund ist stets die Entwicklung von betrieblichen Informationssystemen (kurz: Informationssystem); Software-Entwicklung in anderen Bereichen wie bspw. dem Compilerbau, der Entwicklung von Standardbüroanwendungen (Textverarbeitung, Tabellenkalkulation etc.) oder für Embedded Systems ist hier nicht von Interesse.[22] Ein solches Informationssystem hat stets einen bestimmten Anwendungszweck zu erfüllen. Wissen über diese Anwendungsgebiete wird hier als Domänenwissen bezeichnet.[23] Es sei ergänzt, dass bisher keine zufriedenstellende Systematisierung von Domänenwissen existiert. Beispielsweise ist der Begriff Branche zu unscharf für eine brauchbare Klassifikation und der Begriff Wirtschaftszweig ist ungeeignet.[24]

Allgemein umfasst Technologiewissen Wissen, das Mittel beschreibt, um bestimmte Ziele in einem bestimmten Kontext zu erreichen. Im Kontext der Software-Entwicklung können darunter Verfahren zur Benutzung und Erstellung von Informationssystemen verstanden werden. Einige Technologien werden aufgrund ihrer Omnipräsenz im jeweiligen Anwendungsbereich als Domänenwissen verstanden bzw. können im Einzelfall schwer von Domänenwissen abge-

---

[21] Vgl. Heinrich 1993, S. 103.

[22] Vgl. Denert 1991, S. 15f.

[23] Vgl. Lehner, Hildebrand, Maier 1995, S. 284.

[24] Vgl. Mertens, Lohmann 2000.

grenzt werden, da diese Techniken bereits den Status von Domänenwissen erhalten haben. Beispielsweise kann die Doppelte Buchführung zunächst als eine Technologie zur Abwicklung des Rechnungswesens verstanden werden, die heute wohl eher als Domänenwissen bezeichnet wird, d. h. heute wird praktisch keine andere Technik der Buchführung verwendet. Die Technik der Doppelten Buchführung ist untrennbar mit der Anwendungsdomäne Buchführung verbunden. Ebenso hat das im Grund technisch motivierte MRP-Stufenplanungskonzept von PPS-Systemen inzwischen den Rang von Domänenwissen erhalten.

Projektwissen enthält Wissen zum grundsätzlichen systematischen und planvollen Vorgehen in der Software-Entwicklung. Dieser Bereich zeigt, wie große Projektvorhaben planungstechnisch abzuwickeln sind. Ein zweiter Bereich umfasst allgemeine Technologien zur Risikoabschätzung von Projekten. Der dritte Bereich beschäftigt sich mit der Fragestellung, wie weit ein Projekt aus seinem technisch-sozio-ökonomischen Umfeld herausgelöst und als separate Einheit betrachtet werden kann, ohne wesentliche externe Störgrößen zu vernachlässigen.[25]

Beispiele: 1. Domänenwissen: Die Systemanalytikerin Frau C. erstellt seit 20 Jahren Anforderungsdokumente für Informationssysteme in Handelsunternehmen. Dadurch verfügt sie über ein hohes Wissen in der Domäne der Warenwirtschaftssysteme. 2. Technologiewissen: Der Systemarchitekt Herr S. ist mit dem Funktionsprinzip aktueller Internet-Technologien wie Java-Server-Pages, Java-Applets und Java-Servlets vertraut, so dass er über das notwendige technologische Wissen verfügt, um beurteilen zu können, mit welcher Technologie ein konkretes Projekt der Systementwicklung realisiert werden sollte. 3. Projektwissen: Frau D. ist seit 15 Jahren erfolgreiche Projekt-Managerin von Software-Projekten. Im Laufe ihrer beruflichen Karriere hat sie Projekte in unterschiedlichen Anwendungsbereichen und mit den verschiedensten Technologien erfolgreich durchgeführt. Ihre Erfahrungen in der Projektabwicklung hat sie in den letzten Jahren durch entsprechende quantitative und qualitative Messungen verschiedenster Projektrahmenparameter objektiviert und in einem allgemeinen Lehrbuch zum Projektmanagement in der Software-Entwicklung (Projektwissen) festgehalten.

In Tabelle 1 werden die skizzierten Facetten des Wissensbegriffes aus Sicht der Software-Entwicklung abschließend in Form eines morphologischen Kastens zusammengefasst.

2.2 Bausteine eines Wissensmanagements

In der Literatur existieren unterschiedliche Systematisierungsansätze des WM. Die folgenden Ausführungen basieren auf dem Ansatz von Pawlowsky, der vier Dimensionen unterscheidet:[26]

- *Lernebenen*: Ebenso wie einzelne Menschen einen Lernprozess durchlaufen, kann das gemeinsame Verhalten einer Gruppe von Menschen verschiedene Lernstufen erreichen. Dabei kann das organisationale Lernen nicht bloß auf die Summe der Lerneffekte der einzelnen Mitglieder der Gruppe reduziert werden. Insofern sollte ein WM verschiedene Lernebenen berücksichtigen. Hierbei werden in Anlehnung an die erste Fa-

[25] Vgl. dazu Seiffert, Radnitzky 1992, S. 362f. Es sei darauf hingewiesen, dass die dort vorgenommene allgemeine Klassifikation von Technologien aus wissenschaftstheoretischer Sicht im Kontext der Software-Entwicklung unangemessen erscheint.

[26] Vgl. Pawlowsky 1998, S. 16-22.

cette des Wissensbegriffes (Wissensträger) die Ebenen Individuum, Gruppe, Organisation sowie Netzwerk unterschieden.

- *Lernformen*: Ein WM sollte verschiedene theoretische Formen des Lernens berücksichtigen. Dabei kann zwischen einer Veränderung und Differenzierung individueller kognitiver Strukturen auf der einen Seite, und einer kulturellen Sichtweise, die das Lernen auf einer Ebene der kollektiven Wahrheitsinterpretation und Sinnfrage betrachtet, auf der anderen Seite, unterschieden werden. Als dritte Lernform ist ein verhaltensorientierter Ansatz zu ergänzen.
- *Lerntypen*: Es gibt verschiedene Lerntypen, die als Single-Loop-, Double-Loop- oder Deutero-Learning bezeichnet werden. Betrachtet man Lernprozesse als Regelungs- und Steuerungssysteme, die auf den Aktionsparameter „1. Reflexion, Analyse, Entwicklung von Einsichten", „2. Ziele" und „3. Verhalten" basieren, dann bedeutet ein Single-Loop-Learning das Korrigieren des Verhaltens aufgrund von Abweichungen zwischen den erwarteten Ergebnissen und der wahrgenommenen Realität. Bei einem Double-Loop-Learning werden darüber hinaus ebenso die ursprünglich verfolgten Ziele angepasst. Dagegen bezeichnet der Begriff Deutero-Learning einen Lernprozess, bei dem Ergebnisabweichungen dazu führen, dass die Realität auf eine neue Art und Weise reflektiert und analysiert wird und neue Einsichten über die Wirklichkeit wahrgenommen werden. Für unterschiedliche Aufgabenbereiche kommen verschiedene Lerntypen in Betracht: bspw. können Routine-Abläufe mit einem Single-Loop-Learning erlernt werden.
- *Lernphasen*: Die bisherige Differenzierung hat primär strukturelle Beziehungen eines WM fokussiert. Darüber hinaus kann WM ebenso im Hinblick auf einen Phasenablauf betrachtet werden. Hierbei gilt es, verschiedene Phasen zu unterscheiden und im Hinblick auf unterschiedliche Lernebenen, Lernformen sowie Lerntypen auszugestalten.

Merkmal	**Merkmalsausprägungen**				
Träger	Individuum	Gruppe	Organisation	Netzwerk	Technisches System
Güte	subjektiv			objektiv	
Ausdrücklichkeit	implizit			explizit	
Sprachbezug	nicht-sprachlich		natürlich-sprachlich		formal-sprachlich
Abstraktionsebene	Sachwissen		Konzeptionelles Wissen		Meta-Wissen
Sachbezug	Domänenwissen		Technologiewissen		Projektwissen

Tab. 1: Facettenklassifikation des Begriffes Wissen aus Sicht der Software-Entwicklung

Die verschiedenen Lernphasen des beschriebenen Modells werden im Folgenden einer detaillierteren Betrachtung unterzogen:[27]

- *Identifikation von Wissen*: Ausgangspunkt des Prozesses ist eine systematische Identifikation des relevanten Wissens und seiner Wissensträger innerhalb einer Unternehmung. Darüber hinaus sollten Informations- und Kommunikationsbeziehungen der Unternehmung zum Umfeld erfasst werden. Ferner werden in dieser Phase die verschiedenen Wissensziele definiert.
- *Generierung von Wissen*: Die Erzeugung von neuem Wissen aus den vorhandenen Wissensbeständen sowie der Einsatz weiterer betrieblicher Ressourcen kann als eine Kernfrage des WM verstanden werden. Zur Initiierung derartiger Generierungsprozesse müssen strukturelle, personelle und zeitliche Gestaltungsmöglichkeiten in Betracht gezogen werden. Eine Leitfrage im Prozess der Wissensgenerierung ist die Transformation von implizitem Wissen in explizite Formen. Ebenso ist zu überlegen, inwieweit das Wissen von externen Wissensquellen wie bspw. Lieferanten, Beratern, Hochschulen usw. innerhalb der Organisation genutzt werden kann.
- *Diffusion organisationalen Wissens*: Der Prozess der Wissensverteilung innerhalb der Organisation ist nur dann effizient und effektiv möglich, wenn vorab bekannt ist, welche Wissensbedarfe bei den einzelnen Organisationsteilen existieren. Da es schwierig ist, die einzelnen Diffusionsprozesse auf Basis inhaltlicher Kriterien zu steuern, müssen zunächst entsprechende infrastrukturelle sowie sozialpsychologisch-kulturelle Voraussetzungen geschaffen werden. Dabei sind die verschiedenen Lernebenen zu berücksichtigen: Auf der individuellen Ebene steht die Bereitschaft zur Kommunikation von Wissen sowie die entsprechende Kompetenz im Vordergrund, dagegen stehen auf der Netzwerk-Ebene primär wissenslogistische Aspekte im Fokus der Betrachtung.
- *Integration und Modifikation der Wissensbasis*: Das Handeln von einzelnen Personen, Gruppen und Organisationen basiert auf verschiedenen Vermutungen hinsichtlich Ursache und Wirkung innerhalb der betrieblichen Realität. Innerhalb der Integrations- und Modifikationsphase wird das Ziel verfolgt, die Handlungstheorien zu verstehen, um so entsprechende Veränderungen in den Handlungsschemata der Organisationsmitglieder bei dem Vorhandensein oder Fehlen von Wissen entsprechend steuern zu können. Mit anderen Worten müssen innerhalb dieser Phase die impliziten und expliziten Regeln des Handelns der Organisationsmitglieder reflektiert werden. Dabei ist es besonders wichtig, entsprechende Feedback-Kopplungen im organisatorischen System zu verankern.
- *Aktion und Nutzung*: Kernfrage dieser Phase ist, wie das Verhalten von Organisationsmitgliedern durch die Anwendung von Wissen gestaltet werden kann. Hierbei zeigt sich, dass neues Wissen leicht in neues Verhalten umgesetzt werden kann, wenn Probe- bzw. Trainingshandlungen durchgeführt werden, die beobachtet, analysiert und re-

[27] In der Literatur existieren weitere Ansätze zur Phaseneinteilung eines WM. Schindler 2001, S. 54f., kommt bei einer Untersuchung verschiedener Phasenmodelle zu folgendem Ergebnis: „Die Darstellungen [der Phasen eines WM, die Autoren] variieren zum Teil in der Benennung der einzelnen Prozesse, meinen aber letztlich das gleiche. Einzige Unterscheidungsmerkmale sind die Darstellung der Modelle (so sprechen einige Autoren von Zyklen, um die Permanenz und Dynamik zu betonen) und deren Darstellungstiefe."

flektiert werden. Gleichermaßen relevant ist die Vermeidung von entsprechenden Blockaden für die Verhaltensumsetzung.

3 Ziele eines Wissensmanagements in der Software-Entwicklung

In der Literatur werden eine Reihe von verschiedenen Zielstellungen vorgestellt, die mit einem Wissensmanagement verbunden werden.[28] Allgemein lassen sich diese Zielsetzungen dahingehend zusammengefassen, dass die Ansätze eine systematische und zielgerichtete Identifizierung der Ressource Wissen, ihren Austausch und ihre Verwendung in einer Organisation anstreben. Diese allgemeine Zielsetzung kann im Kontext der Software-Entwicklung hinsichtlich verschiedener Aspekte konkretisiert werden:

- *Effizientes Integrieren von neuen Mitarbeitern in die Gesamtorganisation sowie in laufende Software-Projekte*:
  Das Hinzufügen neuer Mitarbeiter in ein Software-Projekt wird in der Regel zu Reibungsverlusten führen. Diese Friktionen sollten durch WM vermieden werden. Darüber hinaus sollte der zusätzlich entstehende Kommunikationsaufwand durch neue Projektteilnehmer minimiert werden, um das höhere Produktivitätspotential möglichst im vollen Umfang ausschöpfen zu können. Durch ein effizientes WM soll der Effekt des in der Literatur seit über zwei Jahrzehnten bekannten Brooks's Law vermieden werden: „Adding manpower to a late project makes it later."[29] Einem WM muss es gelingen, neue Organisations- bzw. Projektmitglieder effizient in die informationslogistischen Prozesse der Software-Entwicklung zu integrieren.
- *Einführung und Aufrechterhaltung eines Mitarbeiterschulungsprogramms*:
  Der Kenntnisstand, die Erfahrungen und Wissensbedarfe der Mitarbeiter eines Software-Entwicklungsprojektes sind zu harmonisieren. Hierbei sind an ein WM besondere Anforderungen gestellt, da zum einen im Bereich der Software-Entwicklung Technologiewissen wenig beständig ist, und zum anderen kein allgemeiner Konsens darüber besteht, über welche Fähigkeiten ein ausgebildeter Software-Ingenieur verfügen muss. Mit anderen Worten: das notwendige Domänen-, Technologie- sowie Projektwissen ist weder objektiv noch explizit verfügbar. In diesem Zusammenhang sei auf die Anstrengungen der IEEE Computer Society und ACM zur Erstellung eines „Guide to the Software Engineering Body of Knowledge" verwiesen.[30] Ein effizientes WM muss dem Umstand Rechnung tragen, dass einerseits der entsprechende Wissensbedarf der Mitarbeiter gedeckt wird und andererseits Mitarbeiter nur die Schulungsprogramme durchlaufen, die auch tatsächlich ihrem Bedarf entsprechen. Ein weiterer Zweck eines Schulungsprogramms zielt auf die Produktivität von Programmierern ab, da diese um die Faktoren 8:1 bis hin zu 28:1 je nach Aufgabenart und durchgeführter Untersuchung schwanken kann.[31] Derartige Unterschiede sollten dokumentiert werden, da

[28] Vgl. Bullinger, Wörner, Prieto 1998, S. 25; Hasenkamp, Roßbach 1998 S, 958; Rehäuser, Krcmar 1996, S. 18-20.

[29] Brooks 1995, S. 25.

[30] Vgl. o. V. 2001c; Bourque, Dupuis, Abran 1999.

[31] Vgl. Curtis, Hefley, Miller 1995.

sie eine wichtige Grundlage für das Projektmanagement darstellen, darüber hinaus sollte durch geeignete Schulungsprogramme versucht werden, die Produktivität alle Entwickler auf ein hohes Niveau zu heben.

- *Wiederverwendung und Konservierung von Wissen über Projektgrenzen hinweg*:
  Die Ausgestaltung der Prozesse der Software-Entwicklung sowie des WM müssen sicherstellen, dass Wissen, welches im Rahmen eines Entwicklungsprojektes generiert worden ist, problemlos bei anderen Projekten wiederverwendet werden kann bzw. auch anderen Projekten zur Verfügung steht.[32] Hierbei sollten Überlegungen zur Auswahl des favorisierten Entwurfes sowie die analysierten Gründe, warum alternative Entwürfe verworfen worden sind, dokumentiert werden. Diese Überlegung wird von Remme im Kontext der Organisationsgestaltung aufgegriffen.[33] Ferner sollten falsche bzw. unzulängliche Problemlösungen im Entwurf und in der Implementierung dokumentiert und zugänglich gemacht werden, um als Grundlage eines kontinuierlichen organisatorischen Lernprozess dienen zu können. Die Idee, schlechte Problemlösungen systematisch zu dokumentieren, greift die Literatur unter dem Stichwort AntiPattern auf: „An AntiPattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences."[34]

- *Explizierung von Schlüsselwissen*:
  Aus Gründen der Wirtschaftlichkeit sowie der Lernpsychologie erscheint es weder ökonomisch noch rational sinnvoll, prinzipiell alle Wissensbereiche im Prozess der Software-Entwicklung zu explizieren. Allerdings sollte das Wissen, welches zwingend zur erfolgreichen Abwicklung eines Projektes notwendig ist, sprachlich expliziert werden. Andernfalls erscheint das Risiko zu hoch, dass die Träger des Schlüsselwissens das Unternehmen verlassen und damit ein Verlust des entsprechenden Wissens unwiderruflich ist. Diese Zielstellung gewinnt insbesondere vor dem Hintergrund anstehender bzw. geplanter Wartungs- bzw. Pflegeaktivitäten an bestehenden Software-Systemen herausragende Bedeutung.

- *Entlastung der Projektmitglieder von Routine-Aufgaben*:
  Innerhalb der Software-Entwicklung existieren eine Reihe von Informationsprozessen, die prinzipiell hochgradig automatisiert ablaufen können. Dieses Potential sollte systematisch genutzt werden. Hierbei handelt es sich bspw. um automatisierte Build-Prozesse, um das Konfigurationsmanagement oder um Modultests. Diese Prozesse können in der Regel sehr gut informationstechnisch unterstützt werden, wodurch Mitarbeiter entlastet und die Prozessqualität gesteigert werden kann.

- *Vermeidung von externen Störungen des Software-Entwicklungsprozesses*:
  Betrachtet man den Prozess der Software-Entwicklung als einen Prozess der Entwicklung von explizitem und formal-sprachlichem Wissen (Quelltext), kann folgendes Bild von diesem Produktionsprozess gezeichnet werden: Ähnlich wie bei materiellen Produktionsprozessen eine Störung des Produktionsprozesses zu erheblichen Kostenzuschlägen führt, können Wissensverarbeitungsprozesse durch Störungen signifikant negativ beeinflusst werden. Weinberg weist insbesondere auf die negative Wirkung

[32] Vgl. Noth 1987, S. 1f.; Basili, Gianluigi, Rombach 1994.

[33] Vgl. Remme 1997.

[34] Brown et al. 1998, S. 7.

von Störungen bei Review-Sitzungen hin.[35] In einem anderen Zusammenhang betonen DeMarco und Listner den negativen Einfluss auf den Entwicklungsprozess bei Störungen des Entwicklungsteams durch eingehende Telefonanrufe, die zu erheblichen Qualitätseinbußen im Arbeitsprozess führen können.[36] Das Wissensmanagement sollte effiziente Maßnahmen ergreifen, die derartige Störungen verhindern, um so den Prozess der Wissenserzeugung bei der Software-Entwicklung ungestört ablaufen zu lassen.

Die beschriebenen Ziele des WM können im Hinblick auf das (People) Capability Maturity Model[37] (CMM) dahingehend zusammengefasst werden, dass eine Software-Entwicklungsorganisation durch ein effizientes WM darauf abzielt, einen höheren CMM-Reifegrad zu erreichen, indem sämtliche Entwicklungsprozesse aufgrund von vorhandenen Erfahrungen und gewonnenen Erkenntnissen systematisch durchgeführt und kontinuierlich verbessert werden.

4 Aspekte des Wissensmanagements in Modellen der Software-Entwicklung

4.1 Leichtgewichtige versus schwergewichtige Prozesse

In Theorie und Praxis sind seit Beginn der Software-Entwicklung unterschiedliche Prozessmodelle bekannt.[38] In jüngster Zeit werden Vorgehensmodelle in die Kategorien leicht- und schwergewichtig unterteilt:[39]

- *Schwergewichtige Prozesse*: Unified Software Development Process[40], Rational Unified Process[41], V-Modell[42];
- *Leichtgewichtige Prozesse*: Adaptive Software Development[43], Crystal Methodik-Ansätze, insbesondere Crystal Clear[44], Extreme Programming[45], Feature-Driven Development[46], SCRUM[47].

Eine gefestigte Terminologie, die definiert, was unter einem schwer- bzw. leichtgewichtigen Prozess zu verstehen ist, hat sich noch nicht herausgebildet. Martin charakterisiert leichtge-

---

[35] Vgl. Weinberg 1994, S. 165f.

[36] Vgl. DeMarco, Lister 1991, S. 80-86.

[37] Vgl. Curtis, Hefley, Miller 1995; Paulk et al. 1993.

[38] Die Genealogie von Vorgehensmodellen beschreibt Bremer 1998.

[39] Vgl. Coldewey 2001b; Dittmar, Eckstein 2001; Rumpe 2001, S. 121.

[40] Vgl. Jacobson, Booch, Rumbaugh 1998.

[41] Vgl. Rational Software Corporation 2000.

[42] Vgl. Bröhl, Dröschel 1993; o. V. 1997a.

[43] Vgl. Highsmith 2000.

[44] Vgl. o. V. 2001b.

[45] Vgl. Beck 1999; Beck 2000; Beck, Fowler 2000.

[46] Vgl. Coad, Lefebvre, De Luca 1999, S. 182-203.

[47] Vgl. o. V. 2001d, S. 5.

wichtige Prozesse mit dem Attribut „agil“.[48] Intensional werden leichtgewichtige Prozesse unterschiedlich beschrieben: „In einem leichtgewichtigen Prozess wird alles weggeworfen, was seinen Nutzen für das Projekt verloren hat. Dokumente werden nur produziert, wenn sie das Ziel unterstützen, ein System zu entwickeln. Dokumente, die dennoch erstellt werden sollen und die im Wortsinne ausschließlich der Dokumentation dienen, werden wie jedes andere Feature des Systems behandelt. Für die Erstellung der Dokumente wird eine Schätzung abgegeben, die Kosten werden ermittelt, die Beauftragung wird eingeholt und schließlich wird sie in die Iterationsplanung mit aufgenommen.“[49]; Martin beschreibt folgende Prioritäten: „Individuals and interactions over processes and tools“, „Working software over comprehensive documentation“, „Customer collaboration over contract negotiation“, „Responding to change over following a plan“ [50]; Coldewey nennt die Attribute „konsequente Anwendersicht“, „inkrementelles Vorgehen“ sowie „hoher Pragmatismus“[51].

Im Folgenden werden das V-Modell als ein Vertreter schwergewichtiger Prozesse sowie Extreme Programming (XP) als ein Vertreter leichtgewichtiger Prozesse hinsichtlich relevanter Aspekte des WM näher untersucht.

4.2 V-Modell

„Das V-Modell [Vorgehensmodell, die Autoren] ist ein anerkannter Entwicklungsstandard für IT-Systeme, der einheitlich und verbindlich festlegt, was zu tun ist, wie die Aufgaben durchzuführen sind und womit dies zu geschehen hat. Es umfaßt

- das Vorgehensmodell,
- die Methodenzuordnung und
- die funktionalen Werkzeuganforderungen.

Damit ist klar umrissen, in welchen Schritten und mit welchen Methoden die Entwicklungsarbeiten auszuführen sind.“[52] „Das V-Modell wurde ursprünglich im Auftrag des Bundesministeriums für Verteidigung (BMVg) und in Zusammenarbeit mit dem Bundesamt für Wehrtechnik und Beschaffung (BWB) in Koblenz von der Industrieanlagen-Betriebsgesellschaft mbH (IABG*)* in Ottobrunn bei München erstellt. Im Sommer 1992 wurde es vom Bundesministerium des Innern (BMI) für den Bereich der Bundesverwaltung übernommen und ist seit Juni 1996 auch dort eine verbindliche Vorschrift.“[53] Der aktuelle Standard ist das V-Modell ’97, in dem eine Reihe von Ergänzungen und Verbesserungen vorgenommen worden sind (bspw. werden im V-Modell ’97 eine inkrementelle sowie eine objektorientierte Vorgehensweise besser unterstützt). Das V-Modell wird inzwischen nicht nur von Behörden verwendet, sondern hat auch in der Industrie an Verbreitung gewonnen.

Die gesamten Aktivitäten des V-Modell sind in vier Submodelle untergliedert:

---

[48] Vgl. Martin 2001.

[49] Dittmar, Eckstein 2001, S. 29.

[50] Martin 2001, S. 8, siehe auch: o. V. 2001a.

[51] Coldewey 2001a, S. 85.

[52] o. V. 1997b, S. 2.

[53] o. V. 1997b, S. 4.

- Das Projektmanagement (PM) plant, steuert, kontrolliert und informiert die drei Submodelle Systemerstellung (SE), Qualitätssicherung (QS) und Konfigurationsmanagement (KM).
- Das KM verwaltet die Produkte, die im Rahmen der übrigen Aktivitäten erstellt werden. Ein Produkt wird als das Ergebnis einer Aktivität bzw. als Gegenstand einer Aktivität des V-Modells verstanden. Dieses Submodell stellt sicher, dass alle Produkte eindeutig identifiziert werden können, dass verschiedene Versionen eines Produktes unterscheidbar bleiben sowie dass Änderungen an Produkten kontrolliert durchgeführt werden.
- Die QS erarbeitet einerseits entsprechende Qualitätskriterien, Prüfpläne, Prüfkriterien und Standards. Andererseits werden die in den übrigen Modellen entwickelten Produkte den entsprechenden Prüfungen unterzogen.
- Die SE umfasst alle Aktivitäten, die unmittelbar dem Prozess der Software-Entwicklung zuzurechnen sind: System-Anforderungsanalyse, System-Entwurf, Software-Grobentwurf, Software-Feinentwurf, Software-Implementierung, System-Integration u. ä.

Obgleich das V-Modell '97 verschiedene Entwicklungsszenarien wie „Inkrementelle Entwicklung", „Grand Design", „Einsatz von Fertigprodukten", „Objektorientierte Entwicklung", „Software-Pflege und -Änderung" für mögliche zeitliche Abläufe der definierten Aktivitäten vorsieht,[54] kann das V-Modell im Kern als ein um die Qualitätssicherung erweitertes Wasserfall-Modell mit Rückkopplung verstanden werden.[55]

Zur Identifikation von Wissen

Das V-Modell berücksichtigt zwei wesentliche Wissensbereiche. Zum einen geht das V-Modell davon aus, dass prinzipiell alle Ergebnisse eines Projektes in sogenannten Produkten festgehalten werden. Die im Laufe der Entwicklung hervorgebrachten Produkte enthalten alle notwendigen Informationen, die zur Steuerung und Koordination des Entwicklungsprozesses verwendet werden. Die Produkte können als explizites Wissen verstanden werden. Dies bedeutet, dass das V-Modell unterstellt, dass das Wissen im Software-Entwicklungsprozess expliziert werden kann, muss und sollte. Eine informale Kommunikation zwischen den Projektbeteiligten wird von dem Prozessmodell nicht betont. Ferner kann aufgrund der Forderung, von jeder Sitzung bzw. jedem Gesprächstermin ein Protokoll anzufertigen, abgeleitet werden, dass informelle und damit nicht nachvollziehbare Kommunikationsprozesse als unerwünscht angesehen werden. Zu den erstellten Produkten gehören insbesondere auch die Anwenderanforderungen, die aus fachlicher Sicht die Anforderungen an das geplante System zu Beginn der Systementwicklung festschreiben.

Zum anderen beschreibt das V-Modell spezifische Rollen, denen die Ausführung der im Modell definierten Aktivitäten übertragen wird. Beispielsweise sind an der Aktivität „SE 1.5 System fachlich strukturieren" u.a. folgende Rollen beteiligt: Systemanalytiker (verantwortlich), technischer Autor (mitwirkend), Systembetreuer (mitwirkend), Datenadministrator (mitwirkend), IT-Sicherheitsbeauftragter (mitwirkend), IT-Beauftragter (mitwirkend), Anwender

[54] Siehe o. V. 1997a, siehe Teil 3: Handbuchsammlung - Szenarien.

[55] So auch Balzert 1998, S. 101.

(mitwirkend) sowie ein Datenschutzbeauftragter (beratend). Insgesamt werden durch das V-Modell über 20 verschiedene Rollen und damit unterschiedliche Wissensträger identifiziert. An die Inhaber einer Rolle werden spezifische Kenntnisse und Fähigkeiten gestellt, um die Aufgaben fachgerecht durchführen zu können. Über die im Rollenmodell geforderten Kenntnisse und Fähigkeiten können Wissensbedarfe der Inhaber der entsprechenden Rollen abgeleitet werden. Beispielsweise werden „Kenntnis über Methoden und Werkzeuge zur Datenmodellierung“ sowie „Kommunikationsfähigkeit mit Systemanalytiker, Systemdesigner und Anwender“ bei Inhabern der Rolle „Datenadministrator“ vorausgesetzt.

Zur Generierung von Wissen

Die Durchführung einer Aktivität führt zur Erstellung neuer bzw. zur Weiterentwicklung schon vorhandener Produkte. Damit ist die Durchführung einer Aktivität mit einer Zunahme an expliziten Wissen verbunden. Weitere, spezielle Aktivitäten, die darauf abzielen, weitere Wissensbereiche systematisch weiterzuentwickeln, werden im V-Modell nicht explizit eingeführt.

Eine Ausnahme bildet die Aktivität „PM 10: Schulung/Einarbeitung“, die wie folgt definiert wird: „Ziel dieser Aktivität ist die Schulung von Projektmitarbeiten, um ihnen die erforderlichen Kenntnisse für einzelne Arbeitsabschnitte zu vermitteln. Werden vom Projektleiter Defizite im Ausbildungsstand des geplanten Personals festgestellt, so sind entsprechende Aus- und Fortbildungsmaßnahmen zu veranlassen.“[56] Tiefergehende Erläuterungen, wie die Schulungsmaßnahmen durchgeführt werden können, werden nicht gegeben. Darüber hinaus sind für die Aktivität „PM 10“ neben den verantwortlichen „Projektleiter“ und dem mitwirkenden „Projektadministrator“ keine weiteren Rollen beteiligt, die an der Schulung beratend teilnehmen. Ein direkter Austausch von Domänen-, Projekt- oder Methodenwissen zwischen erfahrenen und unerfahrenen Mitarbeitern ist nicht beschrieben. Mit anderen Worten werden im V-Modell keine entsprechende Lern-Prozesse explizit gesteuert und geplant. Stattdessen wird davon ausgegangen, dass der Wissensbedarf eines Mitarbeiters leicht festgestellt und durch ad-hoc-Maßnahmen gedeckt werden kann.

Zusammenfassend lässt sich festhalten, dass das V-Modell grundsätzlich von der Annahme ausgeht, dass neues Wissen im Rahmen der Software-Entwicklung explizit anfällt und der Bedarf an impliziten Wissen durch eine Analyse seitens des Projektmanagers bestimmt werden kann.

Zur Diffusion organisationalen Wissens

Das V-Modell geht davon aus, dass das Wissen zur Entwicklung des geplanten Systems jeweils in entsprechenden Produkten festgehalten wird. Die Kommunikation zwischen den Prozessbeteiligten spielt sich jeweils über die entsprechenden Produkte ab. Dabei ist für jede Aktivität innerhalb des gesamten Prozesses exakt definiert, welche Produkte in welchem Zustand als Eingabe erwartet werden und welche Produkte von einer Aktivität in welchem Zustand als Ausgabe erzeugt werden. Die so definierten Abhängigkeiten zwischen den einzelnen Aktivitäten können als Kommunikationsbeziehungen zwischen den Prozessbeteiligten verstanden werden, über die das in Produkten explizit vorliegende Wissen in der Organisation verteilt wird.

---

[56] o. V. 1997a, Submodell Projektmanagement, S. 1-21.

Damit ein reibungsloser Austausch der Dokumente zwischen den Prozessbeteiligten möglich wird, der darüber hinaus sicherstellt, dass alle Veränderungen an Dokumenten kontrolliert und nachvollziehbar durchgeführt werden, werden umfangreiche Aktivitäten durch das Konfigurationsmanagement (Submodell KM) eingeleitet. Die Prozesse des Konfigurationsmanagements stellen sicher, dass alle Mitarbeiter im Entwicklungsprozess mit den notwendigen Informationen versorgt werden. Indes berücksichtigen die Aktivitäten des Konfigurationsmanagements prinzipiell ausschließlich die Verteilung von Wissen, das von technischen Systemen getragen wird. Die Diffusion von Wissen, das an menschliche Träger gebunden ist, wird nicht betrachtet.

Damit geht das V-Modell von der Annahme aus, dass zu Beginn der Systementwicklung die Informationsbedürfnisse aller Projektteilnehmer bekannt sind und somit die komplette Infrastruktur auf die a priori definierten Kommunikationsbeziehungen ausgerichtet werden kann.

Zur Integration und Modifikation der Wissensbasis

Im Submodell KM wird definiert, dass mindestens die Zustände „geplant“, „in Bearbeitung“, „vorgelegt“ und „akzeptiert“ zur Dokumentation verwendet werden müssen. Jedes Produkt, das im Rahmen des V-Modells entwickelt wird, muss zwingend diese Zustandshistorie durchlaufen. Während der Prüfung wird ein Prüfprotokoll erstellt, das die formalen und inhaltlichen Mängel eines Produktes dokumentiert. Genügt ein Produkt den separat festgelegten Prüfkriterien nicht, wird das Produkt von Seiten der Qualitätssicherung abgewiesen und zu einer erneuten Bearbeitung an den Verantwortlichen zurückgegeben. Durch dieses Verfahren wird ein Regelkreis erzeugt, der dafür sorgt, dass jedes Produkt in der gewünschten Qualität vorliegt.

Die zuvor beschriebenen Regelkreise haben eine unterschiedliche Zykluszeit. Während zum Teil eine sofortige Prüfung des Dokumentes möglich und dadurch ein schnelles Feedback gesichert ist, existieren ebenso Prüfaktivitäten, bei den zwischen dem Zeitpunkt der Erstellung des Dokumentes und der Durchführung der Prüfung mehrere Monate oder sogar Jahre liegen können. Beispielsweise wird nach der SW-Implementierung des Systems eine SW-Integration sowie eine System-Integration vorgenommen, bevor das System in die Nutzung überführt wird. Dabei kann festgestellt werden, dass entsprechende Anforderungen an das System unzureichend berücksichtigt worden sind und daher das fertige System den ehemals erhobenen Anforderungen nicht genügt. Dies führt dazu, dass Änderungen am Produkt „Anwenderanforderungen“ erforderlich werden. Durch derartig lange Zykluszeiten können entsprechende Lernprozesse nur schlecht bei der Anforderungsermittlung stimuliert werden.

Zur Aktion und Nutzung

Die vorherigen Ausführungen haben verdeutlicht, dass zum einen der Wissensbedarf der an der Systementwicklung beteiligten Personen a priori bekannt ist. Zum anderen können entsprechende Wissenslücken durch das Einleiten entsprechender Schulungsaktivitäten, die mehr oder weniger ad-hoc durchgeführt werden, prinzipiell geschlossen werden. Daraus kann die Konsequenz gezogen werden, dass das V-Modell von der Annahme ausgeht, dass das Wissen im Entwicklungsprozess prinzipiell vorhanden ist und von allen Projektteilnehmen auch genutzt werden kann. Es wird weiter davon ausgegangen, dass keine besonderen Maßnahmen ergriffen werden müssen, um das benötigte und vorhandene implizite Wissen in einer Software-Entwicklungsorganisation systematisch nutzbar zu machen. Die Ausgestaltung des Wis-

sensmanagementbausteins Aktion und Nutzung wird daher im V-Modell als rudimentär angesehen.

4.3 Extreme Programming

Extreme Programming (XP) ist eine relativ neue Vorgehensweise zur Software-Entwicklung, die vor allem mit dem Namen von Kent Beck in Verbindung gebracht wird. Beck betont, dass die Ideen von XP weder von ihm selbst entwickelt worden sind noch jede für sich neu ist, sondern dass XP viel mehr eine Kombination der Arbeitsergebnisse einer Reihe von Personen, die Prozesse der Software-Entwicklung bei schnell ändernden Randbedingungen untersucht haben, darstellt.[57] Darüber hinaus hat XP inzwischen eine Reihe von Überarbeitungen und Weiterentwicklungen erfahren, die allerdings nicht konsistent zur Verfügung stehen und teilweise widersprüchlich sind.[58] Daher wird im Folgenden ausschließlich auf Aussagen in gesicherten Quellen[59] zurückgegriffen. Weiterentwicklungen, die in zahlreichen Quellen im Internet verfügbar sind, werden aufgrund ihrer mangelnden Nachvollziehbarkeit und Überprüfbarkeit nicht berücksichtigt, obgleich sie einen Einblick in die überaus dynamische Entwicklungsgeschichte von XP geben können.[60]

Zentrale Zielsetzung von XP ist es, Veränderungen der betrieblichen Realität nicht als ein seltenes Ausnahmeereignis in der Software-Entwicklung zu verstehen, das mit allen Mitteln und Maßnahmen zu bekämpfen ist. Stattdessen werden Veränderungen der Anforderungen an das geplante System als eine wesentliche Randbedingung in der Software-Entwicklung begriffen, denen sich jedes Software-Entwicklungsprojekt und jeder Software-Entwicklungsprozess stellen muss. Daher ist es primäres Ziel von XP, den Prozess der Software-Entwicklung zu flexibilisieren und seine Durchlaufzeiten zu verkürzen, um so ein schnelles Feedback seitens des Kundens zu erhalten, neue Anforderungen aufzunehmen, bekannte zu revidieren bzw. zu verfeinern sowie entsprechende Lernprozesse bei den Projektbeteiligten auszulösen. Dieses Ziel wird mit einer Reihe von Methoden erreicht, die in Tabelle 2 zusammengestellt sind.

Zur Identifikation von Wissen

Die Betrachtung des XP aus Sicht des WM zeigt folgende Wissensbereiche: Die Metaphor (Metapher) beschreibt in einer allgemein verständlichen Sprache, in welcher Art und Weise das System funktioniert. Die Metaphor ist in der Lage, die wesentlichen Funktionen des geplanten Systems sowohl für die Kunden als auch für die Entwickler verständlich zu machen. Es wird darauf Wert gelegt, dass dieses Dokument kurz, prägnant sowie für alle an der Systementwicklung beteiligten Personen verständlich formuliert ist.

Zentrales Element zur Iterationsplanung sind User Stories. Eine User Story beschreibt eine Systemeigenschaft in Prosaform, die für sich genommen eigenständig realisierbar ist sowie für den Kunden einen identifizierbaren Nutzen dargestellt. Jede User Story wird auf einer Kartei-

---

[57] Beck 1999, S. 73. Im diesem Zusammenhang ist insbesondere interessant, dass Beck als Inspirationsquelle u. a. den Artikel „The new new product development game“ (Takeuchi, Nonaka 1986) anführt. Nonaka, Takeuchi 1995, S. 75-78, beschreiben die in diesem Aufsatz genannten Prinzipien als eine wichtige Voraussetzung für ein effizientes WM.

[58] Vgl. Rumpe 2001, S. 121.

[59] Vgl. Beck 1999; Beck 2000; Beck, Fowler 2000.

[60] Siehe dazu Cunningham 2001; Fowler 2001; Jeffries 2001; Wells 2001.

karte notiert. Die Größe der Karteikarte spielt insofern eine Rolle, als keine langatmigen Systembeschreibungen erstellt werden können, sondern ausschließlich die Kernaussagen für ein Szenario festgehalten werden. Dabei sollte eine User Story keine technischen Details enthalten.

Practice	Description
The Planning Game	Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.
Small releases	Put a simple system into production quickly, then release new versions on a very short cycle.
Metaphor	Guide all development with a simple shared story of how the whole system works.
Simple design	The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.
Testing	Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished.
Refactoring	Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.
Pair Programming	All production code is written with two programmers at one machine.
Collective Ownership	Anyone can change any code anywhere in the system at any time.
Continuous integration	Integrate and build the system many times a day, every time a task is completed.
40-hour week	Work no more than 40 hours a week as a rule. Never work overtime a second week in a row.
On-site customer	Include a real, live user on the team, available full-time to answer questions.
Coding standards	Programmers write all code in accordance with rules emphasizing communication through the code.

Tab. 2: Methoden des Extreme Programming, Quelle: Beck 2000, S. 54.

Dem Quelltext wird im XP eine besondere Rolle eingeräumt. Alle Aktivitäten des Entwicklungsprozesses sollten darauf ausgerichtet sein, den Quelltext des geplanten Systems zu entwickeln. Es wird darauf hingewiesen, dass der gesamte Prozess der Systementwicklung letztlich ausschließlich dafür da ist, ablauffähigen Code zu erzeugen. Aus diesem Grunde werden verschiedene Methoden eingeführt, um die Qualität des Codes sicherzustellen. Der Code sollte prinzipiell einfach strukturiert sein und keine Funktionalität enthalten, die im aktuellen Iterationsplan noch nicht vorgesehen ist. Der Quelltext enthält explizites und formales Domänenwissen.

Das im Quelltext darüber hinaus implizit vorhandene Domänenwissen wird durch entsprechende Testfälle expliziert. Systemverhalten bzw. Systemeigenschaften, die nicht durch entsprechende Testfälle nachgewiesen werden können, werden von XP als nicht existent eingestuft. Die Testfälle werden so formuliert, dass sie automatisch ausgeführt werden können. Damit stellen die Testfälle neben den Quelltexten eine weitere Form von explizitem und formalem Domänenwissen dar.

Des Weiteren geht XP von einer Reihe von impliziten Wissensbereichen aus. XP lehnt es ab, zu Beginn der Entwicklung ein vollständiges und umfassendes Anforderungsdokumente für das geplante System zu erstellen. Statt dessen betont XP die Notwendigkeit schneller und kleiner Releases, um bei den Entwicklern ein schrittweise zunehmendes Verständnis für die Anwendungsdomäne zu erzielen. Ferner verdeutlicht der Ansatz des noch näher zu erläuternden Pair Programming, dass bestimmtes Wissen nicht explizit verfügbar ist, weshalb bestimmte Aktivitäten nur im Team durchgeführt werden können, bzw. dass die direkte Einführung von Mitarbeitern in das Projektteam effektiver ist als das Durchführen expliziter Schulungen.

Zur Generierung von Wissen

Der Kommunikation mit dem Kunden wird eine besondere Rolle zugesprochen. Es wird davon ausgegangen, dass die Kundenanforderungen zu Beginn des Projektes nicht vollständig explizit und formal festgehalten werden können, sondern dass hier ein kommunikativer Prozess zwischen Kunden und Auftraggeber geschaffen werden muss. Daher ist es angebracht, dass ein künftiger Benutzer des Systems räumlich und organisatorisch in das Entwicklungsteam integriert wird, weil dieser die tatsächlichen Anforderungen an das geplante Systems am besten kennt. Auf diese Weise können schnelle Antworten hinsichtlich der Interpretation von Kundenanforderungen im Entwicklungsprozess gegeben werden. Es wird davon ausgegangen, dass der Kunde den wesentlichen Input für die Definition von Anforderungen liefert und als Maßstab für die Beurteilung der Projektqualität herangezogen werden kann. Diese Maßnahme kann aus Sicht des WM in der Art interpretiert werden, dass ein externer Wissensträger in die Software-Entwicklungsorganisation integriert wird, um innerhalb der Organisation über zusätzliches Domänenwissen zu verfügen.

Als Planning Game wird ein kooperatives Zusammenspiel einer Gruppe von Personen bezeichnet, das zur Planung des funktionalen Umfangs des nächsten Projekt-Release verwendet wird. Die Teilnehmer an dieser Aktivität haben entweder einen fachlichen oder einen technischen Hintergrund. XP weist explizit darauf hin, dass die Planung in Kooperation zeitgleich durchgeführt wird und nicht eine separate Entscheidung von einem Projektmanager darstellt. Die Verantwortlichkeiten aus fachlicher und technischer Sicht sind geteilt. Aus fachlicher Sicht werden Vorgaben hinsichtlich Umfang und zeitlicher Planung eines Release sowie eine Priorisierung der einzelnen Features eines Release erwartet. In technischer Sicht besteht die Verantwortung für die zeitliche Aufwandsschätzung eines Release, die Bestimmung und Abwägung alternativer Entwicklungsrisiken sowie die detaillierte Ausgestaltung des Entwicklungsprozesses, wie er für den Kontext eines Projektes benötigt wird.[61] Diese rollenspezifische Trennung von Verantwortlichkeiten schlägt sich ebenso in einer personellen Trennung nieder. XP weist darauf hin, dass die Schätzung des Projektaufwandes ein iterativer Prozess ist, der im Rahmen mehrerer Zyklen zu durchlaufen ist, so dass Lerneffekte erzielt werden können.

Darüber hinaus definiert XP die Rolle des Coach, dem folgende Aufgaben eingeräumt werden: „Be available as a development partner, particularly for new programmers beginning to take responsibility or for difficult technical tasks. See long-term refactoring goals, and encourage small-scale refactorings to address parts of these goals. Help programmers with indi-

[61] Vgl. Beck 2000, S. 81-84.

vidual technical skills, like testing, formatting, and refactoring. Explain the process to upper-level-managers."[62] Damit nimmt die Rolle des Coach im Entwicklungsteam eine besondere Stellung ein, da er aus Sicht des WM als ein zentraler Wissensträger verstanden werden kann, der die Basis zur Generierung von neuem Wissen darstellt.

Zur Diffusion organisationalen Wissens

Um ein ausreichendes Technologie- und Domänenwissen bei allen Teammitgliedern sicherzustellen, wird das Prinzip „Collective Code Ownership" verfolgt. Es besagt, dass prinzipiell keine disjunkte Aufteilung des gesamten Quelltextes in einzelne Module vorgenommen wird und damit jedes Modul explizit einem Entwicklungsteam zur ausschließlichen Verantwortung übergeben wird. Stattdessen hat jedes Teammitglied die Erlaubnis, alle Stellen im Quelltext zu verändern, solange die automatisierten Testfälle reibungslos ausgeführt werden können. Auf diese Weise soll sichergestellt werden, dass innerhalb des Entwicklungsteams keine Engpässe aufgrund der Verfügbarkeit einzelner Projektteilnehmer entstehen. Es wird darauf hingewiesen, dass dieses Prinzip nicht bedeutet, dass grundsätzlich alle Projektmitglieder mit allen Systemteilen gleich gut vertraut sind.

Der Begriff Pair Programming umschreibt eine Methode, bei der die Programmierung eines Systemteils, bspw. einer Klasse, von zwei Programmierern gleichzeitig vorgenommen wird. Pair Programming „isn't one person programming while another person watches. Just watching someone program is about as interesting as watching grass die in a desert. Pair programming is a dialog between two people trying to simultaneously program (and analyze and design and test) and understand together how to program better. It is a conversation at many levels, assisted by and focused on a computer."[63] Pair Programming wird mit folgenden Vorteilen verbunden: Zum einen liegt die Produktivität höher als bei der separaten Programmierung und einer anschließenden System-Integration. Zum anderen wird die Code-Qualität verbessert und es werden Erfahrungen bzgl. der Funktionalität einzelner Projektteile ausgetauscht.[64] Das Zitat verdeutlicht, dass das Wissen in Form von Quellcode nicht wie üblich von einer einzelnen Person, sondern in einem 2-Mann-Team erstellt wird.

Pair Programming kann ebenso eingesetzt werden, um neue Mitarbeiter in ein Projekt einzuarbeiten bzw. um das Verständnis für bestimmte Teile des Gesamtsystems zu verbessern. Hierbei wird es möglich, dass erfahrene Mitarbeiter demonstrieren, wie bestimmte Funktionen im System (weiter-)entwickelt werden können. Unerfahrene Mitglieder können auf diese Weise durch Beobachtung ihres Programmierpartners schnell in den Entwicklungsprozess eingegliedert werden. Bei diesem Prozess wird das implizite Wissen von erfahrenen Projektmitgliedern in implizites Wissen von unerfahrenen Projektmitgliedern transformiert, wodurch ein langwieriger und expliziter Schulungsprozess vermieden werden kann.

Pair Programming ist allerdings auch mit Nachteilen verbunden: Beck weist zum einen darauf hin, dass Pair Programming größere Anforderungen an die soziale Kompetenz von Entwicklern stellt und für die meisten Entwickler zunächst ungewöhnlich ist. Allerdings können nach einer gewissen Einarbeitungszeit verschiedene Programmierpaare mit jeweils wechselnden Partnern zu einer hohen Gesamtproduktivität führen. Zum anderen stellt Pair Programming

---

[62] Beck 1999, S. 74.

[63] Beck 2000, S. 100.

[64] Es sei darauf hingewiesen, dass diese Thesen nicht methodisch belegt werden.

ungewöhnliche Anforderungen an die Büromöbelausstattung. Beck erläutert, dass die in vielen Büros anzutreffenden Eckschreibtische für das Pair Programming ungeeignet sind. Vorzuziehen sind rechtwinklige Schreibtische, die bei Bedarf leicht zu größeren Arbeitsflächen zusammengeschoben werden können.[65]

Zur Integration und Modifikation der Wissensbasis

„Nobody wants to talk about testing. Testing is the ugly step-child of software development. The problem is, everybody knows that testing is important. [... Testing in XP, die Autoren] is not the work of someone who loves testing. Quite the contrary. This is the work of someone who loves getting programs working. So you should write the tests that help get programs working and keep programs working."[66] Um dieses Ziel zu erreichen, betrachtet XP den Test-Prozess als integralen Bestandteil des Entwicklungsprozesses, der nicht im Anschluss an die Implementierung durchgeführt werden sollte, sondern mit diesem untrennbar verschmolzen ist. XP geht davon aus, dass durch dieses Prinzip bei den Entwicklern ein starkes Bewusstsein für qualitativ hochwertige Software geschaffen werden kann und Software-Entwickler angespornt werden, Fehler bei der Implementierung zu vermeiden, um so spätere, kostenintensive Nachbesserungen einzusparen.

Zur Steuerung des Prozesses der Systementwicklung werden die zuvor schon angesprochenen Planning Games eingesetzt. Da die Planning Games zyklisch wiederholt werden, erhält der Kunde regelmäßige und realistische Rückmeldungen, in welchem Zustand sich das geplante Software-System zur Zeit befindet. Ferner hat der Kunde stets die Möglichkeit, direkten Einfluss auf die Systementwicklung auszuüben. Er kann nicht nur an wenigen definierten Punkten die weitere Entwicklung eines Projektes bestimmen, sondern kann regelmäßig die Richtung der Systementwicklung vorgeben. Dabei kann der Kunde nicht beliebige Vorgaben an das Entwicklungsteam herantragen. Statt dessen lernt der Kunde, dass jedes zusätzliche Feature im Programm entsprechende Entwicklungszeit erfordert, da er ein unmittelbares Feedback von den Entwicklern über den zeitlichen Aufwand, bestimmte User Stories zu realisieren, erhält. Auf diese Weise wird der Kunde darin unterstützt abzuschätzen, wieviel Aufwand bestimmte Features bei der Implementierung in Anspruch nehmen. Gleichzeitig wird der Kunde gezwungen, eine Priorisierung seiner Wünsche vorzunehmen, da in einem knapp definierten Zeitfenster nicht alle Funktionalitäten realisiert werden können. Die iterative Vorgehensweise erlaubt es dem Kunden, schrittweise den funktionellen Umfang des geplanten Systems zu definieren und Anforderungen an das geplante System kontinuierlich zu verfeinern. Diese Möglichkeit erscheint insbesondere dann reizvoll, wenn das geplante System einen hohen innovativen Charakter hat und der Kunde nicht sinnvoll abschätzen kann, welche Art und in welchem Umfang eine Software-Lösung angestrebt wird. Einschränkend wird von XP darauf hingewiesen, dass dieses Vorgehen nur dann erfolgsversprechend ist, wenn zwischen Auftraggeber und Entwicklungsteam eine entsprechende Vertrauensbasis entstanden ist und das zu erstellende System von überschaubarem Umfang ist.

---

[65] Vgl. Beck 2000, S. 77f.

[66] Beck 2000, S. 115f., vgl. auch Weinberg 1971, S. 5f.

Zur Aktion und Nutzung

„If programming is about learning, and learning is about getting lots of feedback as quickly as possible, then you can learn much from tests written by someone else days or weeks after the code."[67] Ein Software-Entwickler erhält durch das Anstoßen der automatisiert ablaufenden Test-Suite direkte Rückmeldungen, ob eine von ihm vorgenommene Systemveränderung das System weiterhin in einem Zustand belässt, der das ordnungsgemäße Ausführen aller Testfälle gewährleistet. Dieses Feedback gibt einem Entwickler die Möglichkeit, direkt festzustellen, ob die von ihm vorgenommen Systemänderungen oder Erweiterungen nicht nur syntaktisch korrekt sind, sondern ebenso den durch die Testfälle definierten semantischen Anforderungen genügen. Aus Sicht eines WM ist das direkte Feedback als positiv zu bewerten.

Um zu vermeiden, dass die Integration verschiedener Softwarebausteine zu erheblichen Problemen und damit zu erheblichen Kosten führt, fordert XP eine mehrmals täglich durchzuführende Integration aller Systemteile. Bei der Integration wird jeweils ein vollständiger Build-Prozesses durchlaufen.[68] Dieses Vorgehen erlaubt, Integrationsprobleme bei der Systemerstellung frühzeitig zu identifizieren und damit zu beheben. Diese Aktivität kann aus Sicht des WM so interpretiert werden, dass Entwickler zum einen ein permanentes Feedback bezüglich der Integrationsfähigkeit des Gesamtsystems erhalten und zum anderen den Build-Prozess durch regelmäßiges Durchführen trainieren.

4.4 Vergleichende Gesamtbetrachtung

In Abbildung 1 wird der Versuch unternommen, wesentliche Elemente und Abläufe des V-Modells und des XP aus Sicht eines WM graphisch vergleichend gegenüberzustellen. Das V-Modell besteht aus einer festen Menge von Aktivitäten, deren Ablauf exakt definiert sind. Jede Aktivität wird von einer Reihe von Projektbeteiligten durchgeführt, die über das Rollenkonzept bestimmt werden. Rückschritte von einer Aktivität zu einer vorgelagerten werden nur im Fehlerfall akzeptiert bzw. wenn ein Systeminkrement vollständig ausgeliefert wurde. Jede Aktivität erzeugt bzw. bearbeitet eine definierte Menge von Produkten. Für die Erstellung eines Produktes ist genau eine definierte Rolle verantwortlich. Die Durchlaufzeit einer Aktivität ist relativ lang, was durch die Breite des Aktivitätensymbols zum Ausdruck kommt, ebenso ist der inhaltliche Arbeitsumfang einer Aktivität relativ umfassend, was durch die Höhe des Aktivitätensymbols verdeutlicht wird. Das von Kunden verfügbare Domänenwissen stellt zwar einen Input des Entwicklungsprozesses dar, allerdings nimmt es keine herausgehobene Stellung ein.

Im unteren Teil der Abbildung ist der Ablauf des XP schematisch dargestellt. XP kennt zum einen einige Aktivitäten, deren Ausgestaltung vorgegeben sind. Beispielsweise werden die Aktivitäten des Konfigurationsmanagements und des automatischen Testens genau beschrieben. Darüber hinaus existieren weitere Aktivitäten, deren inhaltliche Ausgestaltung nur umrissen wird. Deshalb ist das Aktivitätensymbol in der Abbildung durch eine schematisierte Wolke hinterlegt. Beispielsweise ist die inhaltliche Ausgestaltung der Aktivitäten „Planning Game" bzw. „Pair Programming" schlecht formal greifbar und nicht explizit ausformuliert. Ebenso werden diese Aktivitäten von einem eng zusammenarbeitenden Entwicklungsteam

---

[67] Beck 1999, S. 73f.

[68] Laut Cusumano, Selby 1995, führt die Microsoft Corporation ein tägliches Build bei Softwareprojekten durch, um die Integrationsproblematik zu beherrschen.

durchgeführt, das nicht ausschließlich über formale Produkte kommuniziert. Die Ergebnisse der entsprechenden Aktivitäten lassen sich in zwei Gruppen einteilen. Zum einen entstehen formale Produkte, die in der Abbildung unter dem Begriff Software zusammengefasst werden. Hierbei handelt es sich neben dem fertigen, ablauffähigen Programm ebenso um entsprechende automatisch ablaufende Testfälle. Die Verantwortung für die Software obliegt dem ganzen Projektteam. Ferner berücksichtigen die Methoden des XP das Entstehen weiterer Produkte, die in der Abbildung unter dem Begriff Wissen zusammengefasst worden sind. Hierbei handelt es sich um implizites Projekt-, Methoden- sowie Domänenwissen, das im Verlauf des Entwicklungsprozesses durch Lernprozesse entsteht, allerdings nicht explizit verfügbar gemacht wird. Die Durchlaufzeit eines Entwicklungszyklus beträgt grundsätzlich nur wenige Wochen und der inhaltliche Umfang ist dementsprechend gering. Das Wissen des Kunden wird als ein wesentlicher Input für den Entwicklungsprozess verstanden. Dies wird dadurch betont, dass bei der Durchführung von Entwicklungsaktivitäten ein Vertreter des Kunden im Entwicklungsbüro verfügbar sein sollte.

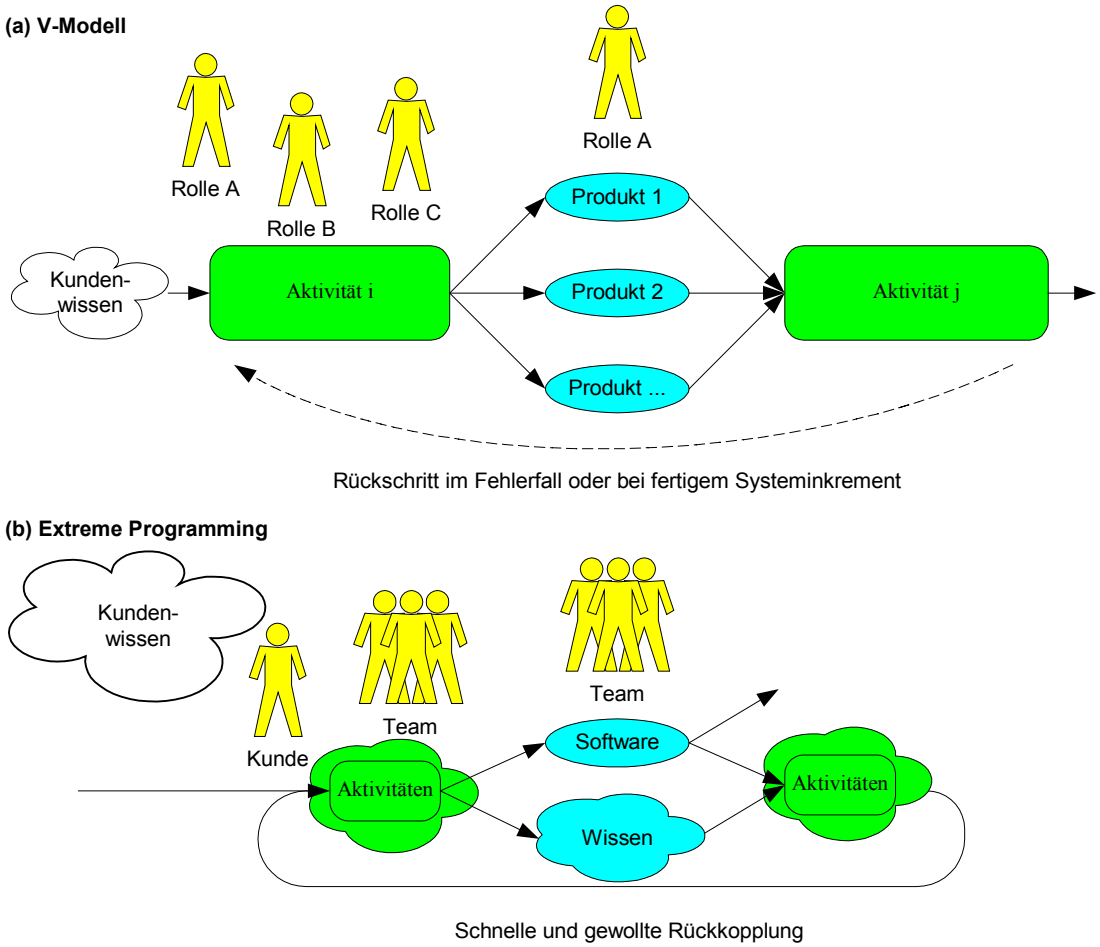


Abb. 1: V-Modell vs. XP aus Sicht des Wissensmanagements

5 Unterstützung eines Wissensmanagements in der Software-Entwicklung

Nachdem zwei Vorgehensmodelle der Software-Entwicklung hinsichtlicht der in ihnen enthaltenen Aspekte des WM untersucht wurden, soll abschließend skizziert werden, wie ein Wissensmanagement in der Software-Entwicklung ausgestaltet werden kann. Im Vordergrund steht dabei die Frage, wie die Wissensmanagementaspekte informationstechnisch unterstützt werden können. Dieser Entwurf kann als Anforderungskatalog verstanden werden.

- *Software-Entwicklungswerkzeuge als Teile des Wissensmanagementsystems*:
  Systeme, die Management, Verwaltung, Erweiterung, Pflege und Nutzung einer Wissensbasis mit Hilfe von Informations- und Kommunikationstechnologien unterstützen, werden unter dem Begriff Wissensmanagementsysteme zusammengefasst.[69] Software-Entwicklungswerkzeuge einer CASE-Umgebung wie Versionsverwaltungs- oder Konfigurationsmanagementsysteme können als Teile eines umfassenden Wissensmanagementsystems betrachtet werden. Da sie bereits State-of-the-Art im Software-Engineering sind, sollen sie jedoch hier nicht weiter vertieft werden.[70]

- *Anlage der Wissensbasis*:
  Das im Laufe des Software-Entwicklungsprozesses explizierte Wissen muss klassifiziert und gespeichert werden. Dies gilt auch für expliziertes Wissen außerhalb des Projektes, das für den Software-Entwicklungsprozess relevant ist. Um das gespeicherte Wissen wiederfinden zu können, muss es in geeigneter Weise zugänglich gemacht werden; nur so kann das Wissen genutzt werden. Solange das explizierte Wissen formalisiert werden kann, können gängige Datenbankmanagementsysteme angewandt werden. Allerdings ist davon auszugehen, dass ein erheblicher Anteil des expliziten Wissens in nicht-strukturierter Form vorliegt bzw. nur mit unverhältnismäßig hohem Aufwand strukturiert werden kann. Für textbasiertes sprachliches Wissen können Techniken von Information-Retrieval-Systemen eingesetzt werden. Beispiele textbasierten Wissens sind die Produkte des V-Modells. Für Bild-, Audio- und Video-Informationen, z. B. als Protokolle von Sitzungen, können multimediale Datenbanksysteme eingesetzt werden. Wichtig ist hierbei die Integration der informationstechnischen Repräsentation des Wissens in der Wissensbasis. Diese kann z. B. mittels Hypertext und Multimedia erreicht werden.[71]

- *Pflege der Wissensbasis*:
  Die Pflege der Wissensbasis ist eine permanente Aufgabe, da ständig neues expliziertes Wissen zugeführt werden muss. Hierzu bedarf es Mechanismen, die es gestatten, in einfacher Weise neues expliziertes Wissen der Wissensbasis zuzuführen. Hierbei stellt sich erstens die Frage, an welcher Stelle im Unternehmen das Wissen der Wissensbasis zugefügt wird, bzw. wer für die Pflege zuständig ist und zweitens, welche Form der Wissensrepräsentation gewählt wird.
  Eine Lösungsmöglichkeit für die erste Fragestellung kann darin bestehen, die Wis-

---

[69] Vgl. Abecker, Decker, Kühn 1998; Frank, Schauer 2001.; Maier, Klosa 1999a, S. 5f.; Maier, Klosa 1999b, S. 4-6

[70] Siehe bspw. Habermann, Leymann 1993.

[71] Siehe dazu Grob, Bensberg 1995; Grob, Bensberg, Bieletzke 1995.

sensakquisition „on the job“ durchzuführen, d. h. der Projektmitarbeiter, bei dem expliziertes Wissen anfällt, pflegt es in die Wissensbasis ein. Der Vorteil liegt darin, dass die Pflege des Wissens unmittelbar erfolgt. Nachteilig ist, dass jeder Mitarbeiter für die Pflege sensibilisiert werden und ein Anreizsystem zur Pflege geschaffen werden muss (ein starkes Anreizsystem stellt bspw. die eigene Nutzung der Wissensbasis für jeden Mitarbeiter dar). Andererseits kann für die Pflege der Wissensbasis eine eigene Rolle definiert werden, ein so genannter Wissensredakteur. Es kann davon ausgegangen werden, dass die Wissensredakteure eine größere Kompetenz für die Struktur der Wissensbasis entwickeln und folglich die Wissensbasis im Laufe der Lebensphase einen homogeneren Aufbau behält. Nachteilig ist jedoch, dass der Wissensredakteur erst Kenntnis über das Vorhandensein von neuem explizitem Wissen gewinnen muss. Daher bietet sich eine Kombination an, bei der die einzelnen Mitarbeiter unmittelbar für die Erfassung zuständig sind, ein Wissensredakteur eine Nachbearbeitung durchführt und das Wissen anschließend in die Wissensbasis einstellt.
Die Repräsentation des expliziten Wissens kann bspw. in Form strukturierter Daten, nicht-strukturierter Texte oder multimedialer Daten erfolgen. Für die Erfassung multimedialer Daten ist besondere Hardware erforderlich (Mikrophone, Kameras, Scanner etc.), die noch nicht an jedem Arbeitsplatz verfügbar ist. Es ist zwar abzusehen, dass durch den Preisverfall derartige Hardware bald ubiquitär wird, doch ist sowohl für multimediale wie für textbasierte Daten eine Klassifizierung, Systematisierung und Aufbereitung (z. B. Verschlagwortung) notwendig. Für derartige Aufgaben bietet sich die oben beschriebene Rolle eines Wissensredakteurs an. Strukturierte Daten in Datenbanken lassen sich zwar prinzipiell einfach pflegen, doch die vorher aufgezeigten Grenzen strukturierter Daten zur Abbildung expliziten Wissens werden auch in Bezug auf die Pflege der Wissensbasis sichtbar: zum Zeitpunkt der Erstellung einer Wissensbasis können nicht die Strukturen von neuem Wissen vorausgesehen werden und es kann somit nur schwer ein längerfristig gültiges Datenbankschema definiert werden.

- *Übersicht über implizites Wissen*:
  Wenn implizites Wissen nicht expliziert werden kann oder soll, so sollte doch die Information darüber, dass implizites Wissen vorhanden ist, allgemein verfügbar sein.[72] Ein Hilfsmittel hierzu sind Wissenslandkarten, mit denen die Träger des impliziten Wissens identifiziert werden können.[73] Zusätzlich können neben den potenziellen Quellen impliziten Wissens auch potenzielle Nutzer oder Bedarfspunkte innerhalb des Entwicklungsprozesses identifiziert werden. So kann die Nutzung des impliziten Wissens gefördert werden.

- *Unterstützung des Austauschs impliziten Wissens*:
  Im Umfeld einer Wissensbasis sollte die Informationstechnik genutzt werden, um die interpersonelle Nutzung impliziten Wissens zu erhöhen. Hierfür bieten sich verschiedene Formen der Groupware-Technologie an.[74] Üblicherweise werden Groupware-Technologien danach unterteilt, ob simultane oder zeitlich versetzte Gruppenprozesse und ob örtlich gebundene oder verteilte Gruppenprozesse unterstützt werden. Nicht-simultane und nicht-strukturierte Gruppenprozesse können beispielsweise durch

[72] Vgl. Heilmann 1999, S. 17.

[73] Vgl. Allweyer 1998.

[74] Vgl. Schwabe, Krcmar 1996.

E-Mail oder Newsgroups unterstützt werden. Hier ist es sinnvoll, dass von derartigen Informationen wechselseitige Bezüge, z. B. in Form von Hyperlinks, zu Artefakten in der Wissensbasis aufgebaut werden können. Zur Unterstützung simultaner Gruppenprozesse in Form von Sitzungen werden Informationstechniken unter dem Begriff CATeam zusammengefasst. Beispielsweise können so bei Brainstorming-, Entscheidungsfindungs- und Abstimmungsprozesse die positiven Effekte gruppendynamischer Sitzungen gestärkt werden. Auch hier sollte eine Schnittstelle zur Wissensbasis aufgebaut werden. So gehören beispielsweise die Argumente einer Design-Entscheidung und der verworfenen Alternativen, die auf einem Whiteboard festgehalten wurden, in die Wissensbasis. Die Nutzung von Groupware erscheint gerade bei XP interessant, da XP besonders auf das implizite Wissen der Mitarbeiter aufbaut und die Notwendigkeit der Dokumentation als zweitrangig betrachtet. So kann Pair-Programming, bei der zwei oder mehrere Personen gleichzeitig an einem Programmcode arbeiten, mit Shared-Editing-Technologien unterstützt werden. Dabei hat jeder Programmierer einen eigenen Bildschirm und eine eigene Tastatur, doch beide bearbeiten gleichzeitig dasselbe Dokument und sehen unmittelbar die Änderungen des Partners. Wird die Arbeitsplatzumgebung zusätzlich mit Audio- und Videoübertragung ausgestattet, so kann das Pair-Programming-Prinzip auch örtlich verteilt durchgeführt werden. Dies ist besonders für die Realisierung des Prinzips des On-site Customer interessant, wenn der Kunde nicht permanent vor Ort sein kann.

- *Wiederverwendung*:
  Die Wiederverwendung bereits entwickelter Analyse-, Entwurfs- sowie Software-Artefakte verspricht eine wesentliche Effizienzsteigerung und Qualitätsverbesserung. Je nach Wiederverwendungsgegenstand wurden unterschiedliche Techniken wie Referenz-Informationsmodelle[75], Patterns[76], Frameworks[77] oder Komponenten[78] entwikkelt. Gemeinsam ist diesen Techniken, dass es sich hierbei um geplante Wiederverwendung handelt. Sie soll die als ungeplant bezeichnete Wiederverwendung ergänzen, die aufgrund des impliziten Wissens von Individuen angewendet wird. Folglich muss die Wiederverwendung in den Software-Entwicklungsprozess integriert werden. Die geplante Wiederverwendung setzt das Explizieren des Wissens um die Artefakte voraus.[79] Das Identifizieren, Aufbereiten, Archivieren und Nutzbarmachen der wiederzuverwendenden Artefakte ist ein Beispiel für Wissensakquisition. Die Verwendung der Artefakte ist eine Form der Nutzung des Wissens.

- *Integration einer Lernumgebung*:
  Aufgrund der Bedeutung des Lernens sollte ein Wissensmanagementsystem Zugriff auf eine Lernumgebung gestatten. Dies kann die Bereiche des Domänen-, des Technologie- sowie des Projektwissens betreffen. Da sich die geplante Integration von Lehrmaterial an den Projektaufgaben und -zielen orientiert, kann vor allem das Single-Loop-Learning unterstützt werden, z. B. das Erlernen einer im Projekt verwendeten

[75] Siehe bspw. Scheer 1997; Becker, Schütte 1996.

[76] Siehe bspw. Gamma et al. 1995.

[77] Siehe bspw. Fayad, Johnson 1999.

[78] Siehe bspw. Turowski 2001.

[79] Vgl. Fettke, Loos 2000, S. 51.

Methode, um diese effizient einsetzen zu können. Aber auch Konzepte des Double-Loop-Learning können im Software-Entwicklungsprozess zur Anwendung kommen. Beispielsweise kann das Erlernen neuer Technologien dazu beitragen, verschiedene Trade-Offs zwischen Technologien bewerten zu können und gegebenenfalls Projektziele anzupassen. Lehrmaterial kann auf diese Weise in Form von Dokumenten eingebracht werden oder als IT-gestützte Lernmodule (E-Learning oder virtuelles Lernen) integriert werden.[80]

- *Einheitlicher Zugang*:
  Es sollte ein einheitlicher Zugang für alle Projektbeteiligten in Form eines Projektportals geschaffen werden. Als Techniken bieten sich Internet-Technologien in Verbindung mit Hypertext und Multimedia sowie einem Web-Browser an. Ein Projektportal stellt eine Untermenge zu einem Unternehmensportal dar. Während Unternehmensportale den Einsteig in die Wissensbasis des gesamten Unternehmens erlauben, beziehen sich Projektportale genau auf ein einzelnes Projekt.[81]

Die vorgenommene Untersuchung der beiden Vorgehensmodelle hat gezeigt, dass aus Sicht eines effizienten WM verschiedene Verbesserungspotentiale bei der Gestaltung der Software-Entwicklungsprozesse vorhanden sind. Die klassische implizite Prämisse, dass das gesamte Wissen im Software-Entwicklungsprozess zu explizieren, zu systematisieren und letztlich zu formalisieren ist, führt zwar zu stabilen Entwicklungsprozessen und zu qualitativ hochwertigen Projektergebnissen. Diese Vorteile werden allerdings erkauft durch verhältnismäßig lange Projektdurchlaufzeiten sowie einem Entwicklungsprozess, der empfindlich auf sich ändernde Systemanforderungen reagiert. Systementwicklung unter den aktuellen wirtschaftlichen Rahmenbedingungen der New Economy bedeutet jedoch nicht, die gegebene betriebliche Realität exakt in Software abzubilden, sondern mit Hilfe von Informations- und Kommunikationssystemen neue Geschäftsprozesse in unbekannten Geschäftsfeldern neu zu gestalten. Dies erfordert eine Vorgehensweise, die den Entwicklungsprozess eines Softwaresystems flexibilisiert und verkürzt. Schwergewichtige Vorgehensmodelle erscheinen vor diesem Hintergrund als zu träge. Leichtgewichtige Vorgehensmodelle wie XP bieten interessante Konzepte, wie Entwicklungsprozesse so gestaltet werden können, dass unsichere und sich stark verändernde Anforderungen und Rahmenparameter effizient antizipiert werden können. Indes sind dem XP bspw. hinsichtlich der Projektgröße und der geographischen Ausdehnung des Projekts enge Grenzen gesetzt. Hierbei erscheint es erfolgsversprechend, die einzelnen Aspekte der unterschiedlichen Ansätze in Form eines Systems zum Wissensmanagement aufzugreifen und gemäß den hier formulierten Anforderungen in den Software-Entwicklungsprozess zu integrieren.

Literaturverzeichnis

Abecker, A.; Decker, S.; Kühn, O. (1998): Organizational Memory. In: Informatik Spektrum 21 (1998), S. 213-214.

Allweyer, T. (1998): Modellbasiertes Wissensmanagement. In: IM Information Management & Consulting 13 (1998) 1, S. 37-45.

[80] Vgl. Ehrenberg et al. 2001.

[81] Vgl. Kappe 2000; Pils 2000; Richter 2000.

Balzert, H. (1998): Lehrbuch der Software-Technik - Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg, Berlin 1998.

Barkow, G.; Hesse, W.; Kittlaus, H.-B.; Luft, A.; Scheschonk, G.; Stülpnagel, A. v. (1989): Begriffliche Grundlagen für die frühen Phasen der Softwareentwicklung. In: IM Information Management & Consulting 4 (1989) 4, S. 54-60.

Basili, V. R.; Gianluigi, C.; Rombach, D. H. (1994): Experience Factory. In: J. J. Marciniak (Hrsg.): Encyclopedia of software engineering. John Wiley & Sons, New York et al. 1994, S. 469-476.

Bea, F. X. (2000): Wissensmanagement. In: WiSt (2000) 7, S. 362-367.

Beck, K. (1999): Embracing Change with Extreme Programming. In: IEEE Computer 32 (1999) 10.

Beck, K. (2000): Extreme Programming Explained - Embrace Change. Addison-Wesley, Boston et al. 2000.

Beck, K.; Fowler, M. (2000): Planning Extreme Programming. 2000.

Becker, J.; Schütte, R. (1996): Handelsinformationssysteme. Landsberg/Lech 1996.

Bourque, P.; Dupuis, R.; Abran, A. (1999): The Guide to the Software Engineering Body of Knowledge. In: IEEE Software 16 (1999) 6, S. 35-44.

Bremer, G. (1998): Genealogie von Entwicklungsschemata. In: R. Kneuper; G. Müller-Luschnat; A. Oberweis (Hrsg.): Vorgehensmodelle für die betriebliche Anwendungsentwicklung. Teubner, Stuttgart, Leipzig 1998.

Bröhl, A.-P.; Dröschel, W. (Hrsg.) (1993): Das V-Modell - Der Standard für die Softwareentwicklung mit Praxisleitfäden. Oldenbourg, München, Wien 1993.

Brooks, F. P. (1995): The Mythical Man-Month - Essays on Software Engineering - Anniversary Edition. Addison-Wesley, Reading, MA, et al. 1995.

Brown, W. J.; Malveau, R. C.; McCormick III, H. W.; Mowbray, T. J. (1998): AntiPatterns - Refactoring Software, Architectures, and Projects in Crisis. John Wiley & Sons, New York et al. 1998.

Bullinger, H.-J.; Wörner, K.; Prieto, J. (1998): Wissensmanagement - Modelle und Strategien für die Praxis. In: H. D. Bürgel (Hrsg.): Wissensmanagement - Schritte zum intelligenten Unternehmen. 1998, S. 21-39.

Coad, P.; Lefebvre, E.; De Luca, J. (1999): Java Modeling in Color with UML - Enterprise Components and Process. Upper Saddle River, NJ, 1999.

Coldewey, J. (2001a): eXtreme Hyping. In: OBJEKTspektrum (2001a) 3, S. 82-85.

Coldewey, J. (2001b): Leichte Prozesse - Motivation und Überblick. In: R. Kneuper; M. Wiemers (Hrsg.): Leichte Vorgehensmodelle - 8. Workshop der Fachgruppe 5.11 der Gesellschaft für Informatik e. V. (GI). Shaker, Aachen 2001b, S. 11-20.

Cunningham, W. (2001): Extreme Programming Roadmap. http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap, Abruf am 2001-07-06.

Curtis, B.; Hefley, W. E.; Miller, S. (1995): People Capability Maturity Model. Software Engineering Institute - Carnegie Mellon University. Pittsburgh, Pennsylvania 1995.

Cusumano, M. A.; Selby, R. W. (1995): Microsoft secrets: how the world's most powerful software company creates technology, shapes markets, and manages people. Free Press, New York et al. 1995.

DeMarco, T.; Lister, T. (1991): Wien wartet auf Dich! Der Faktor Mensch im DV-Management. Hanser, München, Wien 1991.

Denert, E. (1991): Software-Engineering - Methodische Projektabwicklung. Springer, Berlin et al. 1991.

Dittmar, T.; Eckstein, J. (2001): "Was bin ich froh, dass ich ein dünner Hering bin" - Eine Auseinandersetzung mit XP und anderen leichtgewichtigen Methoden. In: OBJEKTspektrum (2001) 1, S. 28-30.

Ehrenberg, D.; Scheer, A.-W.; Schumann, M.; Winand, U. (2001): Implementierung von interuniversitären Lehr- und Lernkooperationnen: Das Beispiel WINFOLine. In: Wirtschaftsinformatik 43 (2001) 1, S. 5-11.

Eulgem, S. (1998): Die Nutzung des unternehmensinternen Wissens - Ein Beitrag aus der Perspektive der Wirtschaftsinformatik. Peter Lang, Frankfurt am Main et al. 1998.

Fayad, M. E.; Johnson, R. E. (1999): Domain-Specific Application Frameworks. Wiley Computer, New York et al. 1999.

Fettke, P.; Loos, P. (2000): Komponentendokumentationen - Eine systematische Bewertung von Ordnungssystemen aus formaler Sicht. In: K. Turowski (Hrsg.): Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme, Siegen, Deutschland, 12. Oktober 2000, Tagungsband. Siegen 2000, S. 51-70.

Fowler, M. (2001): XP and Agile Methods. http://martinfowler.com/articles.html#N116, Abruf am 2001-07-06.

Frank, U.; Schauer, H. (2001): Software für das Wissensmanagement. In: WISU 30 (2001) 5, S. 718-726.

Friedewald, M.; Rombach, D. H.; Stahl, P.; Broy, M.; Hartkopf, S.; Kimpeler, S.; Kohler, K.; Wucher, R.; Zoche, P. (2001): Softwareentwicklung in Deutschland - Eine Bestandsaufnahme. In: Informatik Spektrum 24 (2001) 2, S. 81-90.

Gamma, E. (1992): Objektorientierte Software-Entwicklung am Beispiel von ET++ - Design-Muster, Klassenbibliothek, Werkzeuge. Springer, Berlin et al. 1992.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995): Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA et al. 1995.

Gemünden, H. G. (1993): Information: Bedarf, Analyse und Verhalten. In: W. Wittmann; W. Kern; R. Köhler; H.-U. Küppler; K. v. Wysocki (Hrsg.): Handwörterbuch der Betriebswirtschaftslehre. Bd. 2, Schäffer-Poeschel, Stuttgart 1993, S. 1725-1735.

Grob, H. L.; Bensberg, F. (1995): Multimedia. Arbeitsbericht Nr. 3. Westfälische Wilhelms-Universität Münster - Institut für Wirtschaftsinformatik - Lehrstuhl für Wirtschaftsinformatik und Controlling. http://www.wi.uni-muenster.de/aw/calcat/ab3/index.htm, Abruf am 2001-07-06.

Grob, H. L.; Bensberg, F.; Bieletzke, S. (1995): Hypertext. Arbeitsbericht Nr. 4. Westfälische Wilhelms-Universität Münster - Institut für Wirtschaftsinformatik - Lehrstuhl für Wirtschaftsinformatik und Controlling. http://www.wi.uni-muenster.de/aw/calcat/ab4/index.htm, Abruf am 2001-07-06.

Habermann, H.-J.; Leymann, F. (1993): Repository - Eine Einführung. Oldenbourg, München, Wien 1993.

Hasenkamp, U.; Roßbach, P. (1998): Wissensmanagement. In: WISU 27. (1998) 8-9, S. 956-964.

Heilmann, H. (1999): Wissensmanagement - ein neues Paradigma? In: HMD (1999) 208, S. 7-23.

Heinrich, L. J. (1993): Wirtschaftsinformatik - Einführung und Grundlegung. Oldenbourg, München, Wien 1993.

Heinrich, L. J.; Roithmayr, F. (1995): Wirtschaftsinformatik-Lexikon. 5. Aufl., Oldenbourg, München, Wien 1995.

Hesse, W.; Barkow, G.; Braun, H. v.; Kittlaus, H.-B.; Scheschonk, G. (1994): Terminologie der Softwaretechnik - Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen - Teil 1: Begriffssystematik und Grundbegriffe. In: Informatik Spektrum 17 (1994), S. 39-47.

Highsmith, J. A. (2000): Adaptive Software Development - A Collaborative Approach to Managing Complex Systems. Dorset House, New York 2000.

Huitt, W. (2000): Bloom et al.'s Taxonomy of the Cognitive Domain. http://chiron.valdosta.edu/whuitt/col/cogsys/bloom.html, Abruf am 2001-07-06.

Igl, G.; Lehner, F. (2000): Wissensmanagement in der Beratungsbranche. Forschungsbericht Nr. 39. Universität Regensburg - Lehrstuhl für Wirtschaftsinformatik III. Regensburg 2000.

Jacobson, I.; Booch, G.; Rumbaugh, J. (1998): The Unified Software Development Process. Addison-Wesley, Reading, MA 1998.

Jeffries, R. (2001): XProgramming.com. http://www.xprogramming.org/, Abruf am 2001-07-06.

Kappe, F. (2000): Informations-Portale für unternehmensweites Wissensmanagement. In: IM Information Management & Consulting 15 (2000) 2, S. 36-39.

Kurbel, K. (1992): Entwicklung und Einsatz von Expertensystemen - Eine anwendungsorientierte Einführung in wissensbasierte Systeme. 2. Aufl., Springer, Berlin et al. 1992.

Lehner, F.; Hildebrand, K.; Maier, R. (1995): Wirtschaftsinformatik - Theoretische Grundlagen. Hanser, München, Wien 1995.

Maier, R. K.; Klosa, O. W. (1999a): Knowledge Management Systems '99: State-of-the-Art of the Use of Knowledge Management Systems. Forschungsbericht Nr. 35. Universität Regensburg - Lehrstuhl für Wirtschaftsinformatik III. Regensburg 1999a.

Maier, R. K.; Klosa, O. W. (1999b): Wissensmanagementsysteme - Begriffsbestimmung, Funktionen, Klassifikation und Online-Marktüberblick. Forschungsbericht Nr. 36. Universität Regensburg - Lehrstuhl für Wirtschaftsinformatik III. Regensburg 1999b.

Martin, R. C. (2001): Agile Development: Principles, Patterns, and Process. http://www.objectmentor.com/publications/agileProcess.pdf, Abruf am 2001-07-06.

Mertens, P.; Lohmann, M. (2000): Branche oder Betriebstyp als Klassifikationskriterien für die Standardsoftware der Zukunft? - Erste Überlegungen, wie künftig betriebswirtschaftliche Standardsoftware entstehen könnte. In: F. Bodendorf; M. Grauer (Hrsg.): Verbundtagung Wirtschaftsinformatik 2000. Shaker, Aachen 2000, S. 110-136.

Nonaka, I.; Takeuchi, H. (1995): The Knowledge-Creating Company. Oxford University Press, New York, Oxford 1995.

Noth, T. (1987): Unterstützung des Managements von Software-Projekten durch eine Erfahrungsdatenbank. Springer, Berlin et al. 1987.

o. V. (1997a): Das V-Modell - Allgemeiner Umdruck Nr. 250: Vorgehensmodell - Planung und Durchführung von IT-Vorhaben - Entwicklungsstandard für IT-Systeme des Bundes, Teil 1: Regelungsteil, Teil 3: Handbuchsammlung. 1997a.

o. V. (1997b): Das V-Modell - Entwicklungsstandard für IT-Systeme des Bundes - Kurzbeschreibung. 1997b.

o. V. (2001a): The Agile Manifesto. http://agilealliance.org/, Abruf am 2001-06-01.

o. V. (2001b): Crystal software development methodologies. http://www.crystalmethodologies.org/, Abruf am 2001-07-02.

o. V. (2001c): Guide to the Software Engineering Body of Knowledge. http://www.swebok.org/, Abruf am 2001-07-06.

o. V. (2001d): SCRUM. http://www.controlchaos.com/, Abruf am 2001-07-02.

Paulk, M., C.; Curtis, B.; Chrissis, M. B.; Weber, C. V. (1993): Capability Maturity Model for Software, Version 1.1. Software Engineering Institute - Carnegie Mellon University. Pittsburgh, Pennsylvania 1993.

Pawlowsky, P. (1998): Integratives Wissensmanagement. In: P. Pawlowsky (Hrsg.): Wissensmanagement - Erfahrungen und Perspektiven. Gabler, Wiesbaden 1998, S. 9-45.

Pils, J. (2000): Internet-Portale als Basis im Informationsdschungel. In: IM Information Management & Consulting 15 (2000) 2, S. 15-17.

Rational Software Corporation (Hrsg.) (2000): Rational Unified Porcess. CD ROM. 2000.

Raub, S. (2000): Wissensmanagement. In: Gabler (Hrsg.): Gabler Wirtschafts-Lexikon. 15. Aufl., Gabler, Wiesbaden 2000, S. 3543-3546.

Rehäuser, J.; Krcmar, H. (1996): Wissensmanagement im Unternehmen. In: G. Schreyögg; P. Conrad (Hrsg.): Managementforschung 6 - Wissensmanagement. Walter de Gruyter, Berlin, New York 1996, S. 1-40.

Remme, M. (1997): Konstruktion von Geschäftsprozesse - Ein modellgestützter Ansatz durch Montage generischer Prozeßpartikel. Gabler, Wiesbaden 1997.

Richter, M. (2000): Portal-Technologie als Werkzeug für Wissensmanagement. In: IM Information Management & Consulting 15 (2000) 2, S. 46-50.

Rumpe, B. (2001): Extreme Programming - Back to Basics? In: G. Engels; A. Oberweis; A. Zündorf (Hrsg.): Modellierung 2001 - Workshop der Gesellschaft für Informatik e. V. (GI), 28.-30. März 2001 in Bad Lippspringe. Bd. GI-Edition - Lecture Notes in Informatics (LNI) - Proceedings, Springer, Paderborn, Frankfurt 2001, S. 121-132.

Scheer, A.-W. (1997): Wirtschaftsinformatik - Referenzmodelle für industrielle Geschäftsprozesse. 7. Aufl., Springer, Berlin et al. 1997.

Scheer, A.-W. (1998): Wissen ist Macht. In: IM Information Management & Consulting 13 (1998) 1, S. 3.

Schindler, M. (2001): Wissensmanagement in der Projektabwicklung - Grundlagen, Determinanten und Gestaltungskonzepte eines ganzheitlichen Projektwissensmanagements. 2. Aufl., Josef Eul, Lohmar, Köln 2001.

Schwabe, G.; Krcmar, H. (1996): CSCW-Werkzeuge. In: Wirtschaftsinformatik 38 (1996) 2, S. 209-224.

Seidel, S.; Lehner, F. (2000): Wissensmanagement: Begriffsauffassung und Umsetzung in der Praxis. Analyse auf der Basis von Literaturberichten. Forschungsbericht Nr. 40. Universität Regensburg - Lehrstuhl für Wirtschaftsinformatik III. Regensburg 2000.

Seiffert, H.; Radnitzky, G. (Hrsg.) (1992): Handlexikon der Wissenschaftstheorie. dtv, München 1992.

Solingen, R. v.; Bergout, E.; Kusters, R.; Trienekans, J. (2000): From process improvement to people improvement: enabling learning in software development. In: Information and Software Technology 42 (2000), S. 965-971.

Takeuchi, H.; Nonaka, I. (1986): The new new product development game. In: Harvard Business Review (1986) January-February, S. 137-148.

Turowski, K. (2001): Fachkomponenten - Komponentenbasierte betriebliche Anwendungssysteme. Magdeburg 2001.

Weinberg, G., M. (1971): The Psychology of Computer Programming. Van Nostrand-Reinhold, New York 1971.

Weinberg, G., M. (1994): Systemdenken und Softwarequalität. Hanser, München, Wien 1994.

Wells, J. D. (2001): Extreme Programming: A gentle introduction. http://www.extremeprogramming.org/, Abruf am 2001-07-06.

Willke, H. (1998): Systematisches Wissensmanagement. Lucius & Lucius, Stuttgart 1998.

Working Papers of the Research Group Information Systems & Management:

Paper 1: Fettke, P.; Loos, P.; Thießen, F.; Zwicker, J.: Modell eines virtuellen Finanzdienstleisters: Der Forschungsprototyp cofis.net 1. April 2001.

Paper 2: Loos, P.; Fettke, P.: Aspekte des Wissensmanagements in der Software-Entwicklung am Beispiel von V-Modell und Extreme Programming. Juli 2001.